École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

Institute of Mathematics
Chair of Statistical Data Science

Edge-centric Explainability Methods for Graph Neural Networks in a Regression Setting

— MATH-498 | Semester project (Spring 2023)

By:
Eliott Van Dieren

Supervisors:
Charles Dufour
Prof. Sofia Olhede

June 2023

Contents

1 Introduction

Modelling data using graphs has increasingly proven valuable in multiple real-world applications such as social networks [23], transport [11], chemistry [10] and telecommunication [13]. This is attributed to their ability to effectively capture and represent complex relationships in the data. However, some difficulties arise when working with graphs, as they represent intertwined relational structures, and each of their features may contain its own information. Additionally, the non-Euclidean nature[1] of these mathematical objects implies that there are no familiar properties such as a common system of coordinates or vector space structure, as highlighted in [3]. To address this challenge and leverage the rich graph information, graph neural networks (GNNs) have arisen as a class of deep neural networks (DNNs) well suited for tasks involving graphs. More precisely, their power lies in their ability to propagate information from neighbouring nodes, effectively capturing both the inherent graph structure and the node attributes [24].

However, DNNs are machine learning models renowned for having a "black box" nature, as they lack transparency in explaining their prediction in a human-understandable way. Consequently, this makes them less appealing for critical decision-making and high-risk scientific fields such as medical imaging [5, 14] and drug discovery [7, 22]. GNNs make no exception in this regard, as conveying information across a graph can become quite sophisticated. Therefore, explaining their predictions is still a subject of ongoing research, and substantial effort is currently put into enhancing GNN explainability[2].

For GNNs, most of the work on explainability methods has targetted classification tasks [1, 15, 16, 24]. The regression setting is currently overlooked but remains highly valuable for numerous applications such as materials science [18], chemistry [10, 12] and cognitive score prediction [8].

The contribution of this work can be summarized as follows: first, we adapted existing explainability methods from the classification domain to cater to regression tasks, hence extending their applications to this new context and addressing the lack of methods in this context. Second, we modified classification metrics from [1] to effectively evaluate the performance of these regression-based explainers. The study focuses on three explainability methods: the GNNExplainer [24], the EdgeSHAPer [16] and a modified version of the latter, referred to as the regSHAPer, which is introduced in this work. Through this exploration, this study has provided valuable insights into GNN explanations for regression tasks, and yielded preliminary findings that shed light on the performance of the regSHAPer in comparison to the two other explainers.

Contentwise, Section 2 provides a formal definition of graph and GNNs. Section 3 delves into the field of GNN explanation from a classification task perspective. It covers important explaination concepts, and describes the four categories of explainability methods for graphs and existing metrics to assess the performance of the methods. In Section 4, two existing explainers are described: GNNExplainer [24] and EdgeSHAPer [16]. While work from [6] contributed to the usage of the GNNExplainer for regression, the EdgeSHAPer has been created for classification only, and will need to be modified accordingly. Section 5 presents the contributions and modifications made to transition from the classification to the regression setting. This includes discussions on modifications to the EdgeSHAPer algorithm, the introduction of the regSHAPer method, and adjustments to the previously described metrics. Section 6 conducts a comparative analysis of the three explainability methods using the regression-modified metrics and algorithms described in Section 5 on a real-world quantum chemistry dataset. Lastly, we conclude in Section 7 with a summary of the work, and future directions of this project.

2 Graph neural networks

This section covers multiple terms, notations and theoretical concepts, which will be used extensively throughout this work. We start by recalling graph theory concepts in Section 2.1, while Section 2.2 describes important concepts regarding the foundation layers of GNNs.

[1] A data that has a non-Euclidean structure can not easily be mapped to $\mathbb{R}^n$ (for some dimension n). For instance, a graph can have its physical structure represented in $\mathbb{R}^3$ but this will cause a loss of information (e.g. node/edge attributes).
[2] For the definition of explainability and difference with interpretability, we refer to the beginning of Section 3.

2.1 Graphs

Graph A graph $G(V,E) \in \mathcal{G}$ is a collection of mathematical objects called nodes V with $|V| = N$ and edges $E \subseteq V \times V$. The edges can be seen as links, such that for $u, v \in V$, an edge of G can be denoted by $\{u, v\} \in E$[3]. For $u, v \in V$, we say that u is adjacent to v if $\{u, v\} \in E$. Additionally, the graph G contains self-loops if $\exists u \in V$ such that $\{u, u\} \in E$.

Directed graph A graph is said to be directed if its edges have a direction. Let $H(V', E')$ be a directed graph, and $u, v \in V'$, there exists an edge from u to v if $\{u, v\} \in E'$, however there might not be an edge from v to u, namely $\{v, u\} \notin E'$.

Simple graph A simple graph is a graph $G(V,E)$ such that $\forall u \in V, \{u, u\} \notin E$ (no self-loops), and that there exists only one edge connecting any pair of nodes in G.

Node degree Given a graph $G(V,E)$, we further define the degree of $u \in V$, $\deg(u)$ as the number of edges incident to u. If a given node has a degree 0, then it is called isolated. For directed graphs, the indegree of a node is the number of edges leading into that node and its outdegree, the number of edges leading away from it.

Edge density Given a simple undirected graph $G(V,E)$, the edge density of G $\rho(G)$ is defined as the ratio between the number of edges in G, and the maximum number of edges in G:

$$\rho(G) = \frac{2|E|}{N(N-1)},$$

where $|\cdot|$ refers to the cardinality of a set. There is at most $N(N-1)/2$ edges in an undirected graph, as each node can only be connected once to every other node but himself. For information, one calls those fully-connected graphs *complete* graphs, which are simple undirected graphs in which every pair of nodes are adjacent.

Adjacency matrix To represent the adjacency of nodes of a graph $G(V,E)$, we define the adjacency matrix as a $N \times N$ matrix A such that $\forall 1 \leq i, j \leq N$,

$$A_{i,j} = \begin{cases} 1 & \text{if } \{u_i, u_j\} \in E \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

Undirected graphs will have a symmetric adjacency matrix A, while directed graphs do not necessarily. Furthermore, there exists another definition of the adjacency matrix where one sets $A_{i,i} = 2$ if $\{u_i, u_i\} \in E$ for some properties (not discussed here).

Connectivity Two nodes u, v of a graph $G(V,E)$ are connected if there exists at least one sequence of adjacent nodes, starting from u which leads to v. For a graph $G(V,E)$ to be connected, every pair of nodes $(u, v) \in V \times V$ must be connected.

Neighborhood The neighborhood of a given node $u \in V$ of a graph $G(V,E)$ is defined as

$$\mathcal{N}(u) = \{v : \{u, v\} \in E\},$$

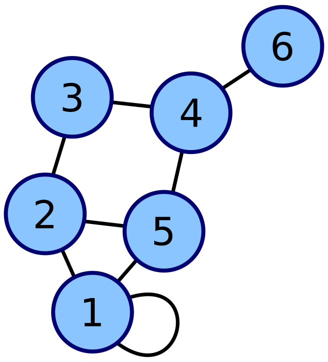
namely the set of adjacent nodes to u. For simple undirected graphs, $|\mathcal{N}(u)| = \deg(u)$, as the number of neighbours will be equal to its number of edges.

Complement graph The complement graph of a simple graph $G(V,E)$ is a graph $\bar{G}(V_C, E_C)$ where

$$E_C = K \setminus E, \quad V_C = V,$$

where K consists of all 2-element subsets of V. For a directed graph $G(V,E)$, it follows the same definition except K now being the 2-element subsets of V, with edges in both directions.

[3] We will assume here that there are no multi-edges.



$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Figure 1: Example of an undirected graph with 6 nodes, 8 edges including one self-loop (left) and its adjacency matrix (right).

Subgraphs A subgraph of $G(V, E)$ is a graph $G_S(V', E')$ satisfying $V' \subseteq V$ and $E' \subseteq (V' \times V') \cap E$. The subgraph $G_{S,u} = G - u$ is defined by $V'_u = V \setminus \{u\}$ and

$$E'_u = E \setminus \{e : e \text{ is indicent to } u\}.$$

Analogously, the subgraph $G_{S,e} = G - e$ is defined by $E'_e = E \setminus \{e\}$ and $V'_e = V$.

Extension with attributes One can extend the graph definition with feature attributes, namely $G(V, E, V_{attr}, E_{attr}) \in \mathcal{G}$. Each node and edge can contain attributes, which are respectively stored in $V_{attr} \subseteq \mathbb{R}^{N \times d}$, for d-dimensional node features and $E_{attr} \subseteq \mathbb{R}^{|E| \times d'}$ for d'-dimensional edge features [4]. In this work, we will often write $G \in \mathcal{G}$, or simply $G(V, E)$ as shortened version of $G(V, E, V_{attr}, E_{attr})$.

2.2 Graph neural networks

Let a parameterized function

$$\begin{aligned} M : \quad & \mathcal{G} \to \mathbb{R}^k \\ & G \mapsto \mathbf{o}, \end{aligned} \tag{2.2}$$

where $G \in \mathcal{G}$ a graph which can be directed or undirected, with or without self-loops, and $\mathbf{o} \in R^k$, the output of the model. One considers M to be a graph neural network if it implements two fundamental layers [4] :

1. **Permutation invariant layer** :

$$h_{t+1}(u_i) = \phi\left(h_t(u_i), \bigoplus_{u_j \in \mathcal{N}(u_i)} a(u_i, u_j)\psi(h_t(u_i), h_t(u_j))\right) \tag{2.3}$$

where $h_{t+1}(u_i)$ represents the attributes values of node u_i after the t^{th} iteration. $\bigoplus$ is a permutation-invariant function which *aggregates* attributes from the neighborhood of node u_i, named $\mathcal{N}(u_i)$. The attributes from nodes in $\mathcal{N}(u_i)$ can potentially be transformed by ψ, sometimes using $h_t(u_i)$ itself. We denote by $a(u_i, u_j)$ the weight of the contribution of node u_j. Lastly, the function ϕ is the *update* function which updates the attributes values of node u_i once the aggregation and transformation steps are finished.

For example, one can choose $\psi(h_t(u_i), h_t(u_j)) = h_t(u_j)$, $\bigoplus$ the sum operator, and ϕ the mean of the attribute of the node with the attributes of its neighbours, with $a(u_i, u_j) = 1, \forall u_j \in \mathcal{N}(u_i)$. This yields the following permutation invariant layer

$$h_{t+1}(u_i) = \frac{1}{\deg(u_i) + 1}\big(h_t(u_i) + \sum_{u_j \in \mathcal{N}(u_i)} h_t(u_j)\big).$$

In Figure 2, one can visualize an iteration of the attribute update process for a given node.

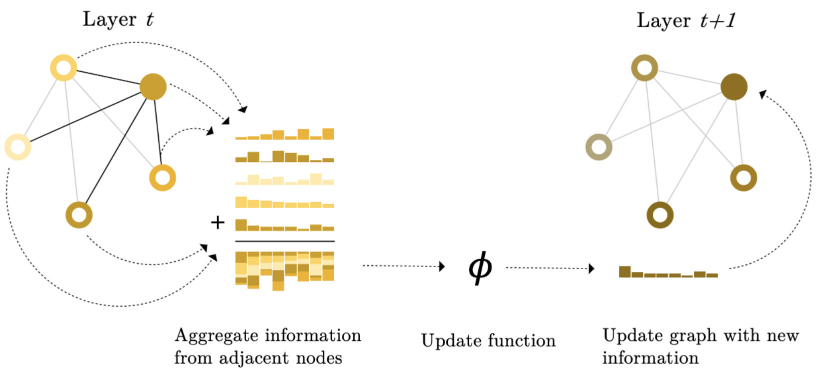

Layer t
Layer t+1
+
ϕ
Aggregate information from adjacent nodes
Update function
Update graph with new information

Figure 2: Example of an iteration of the attribute update for a given node from [19] (modified). Here, the $\bigoplus$ operator is the sum operator, and $a(u_i, u_j) = 1$. This iteration step is done for each node of the input graph, and repeated L number of type (for some L number of hidden layers in the GNN).

2. **Readout layer**: a global pooling layer, which makes the last aggregation before returning the output.

As seen in Figure 3, the function M will therefore take a graph $G(V, E)$ as input and apply the permutation invariant layer for each feature i for L iterations (number of layers in the GNN). Those node attributes can also be used to update edge attributes, as seen by the colour difference of the transformed graph in the figure. Lastly, it will use the readout layer on the transformed graph to output the prediction.

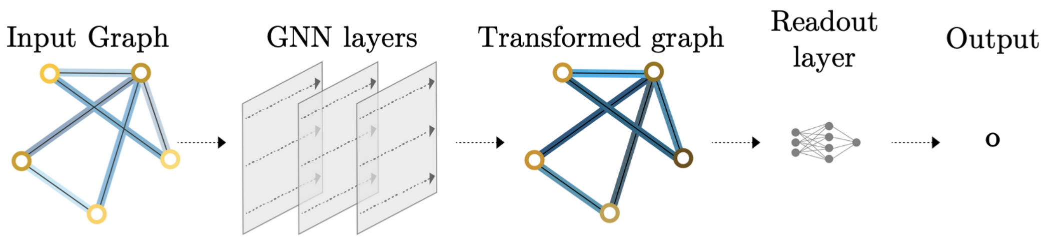


Figure 3: End-to-end prediction task with a GNN model from [19] (modified).

As the geometric deep learning field is a fast-evolving research topic, it can be challenging to properly define GNN layers. Fortunately, the majority of the literature may be derived from three types of layers: convolutional, attentional and message-passing, which all implement the three functions described above [4]. Each of those layers implements the permutation invariant layer differently, as described hereunder:

1. **Convolutional** : the features of $\mathcal{N}(u_i)$ are directly aggregated with $a(u_i, u_j) = c_{i,j}$ fixed for every pair of nodes u_i, u_j, which specify the importance of u_j to the node u_i. Graph convolutional networks are based on this permutation invariant layer. Mathematically, it yields:

$$h_{t+1}(u_i) = \phi\left(h_t(u_i), \bigoplus_{u_j \in \mathcal{N}(u_i)} c_{i,j}\psi_c(h_t(u_j))\right) \tag{2.4}$$

2. **Attentional**: in this case, $a(\cdot, \cdot)$ becomes a *learnable self-attention mechanism* (see [21] for more details) which computes the importance weights of neighbours implicitly. This means that $a(u_i, u_j)$ now depends on the attribute values of both nodes u_i, u_j, and is therefore not fixed anymore.

Mathematically, it yields :

$$h_{t+1}(u_i) = \phi\left(h_t(u_i), \bigoplus_{u_j \in \mathcal{N}(u_i)} a(u_i, u_j)\psi_a(h_t(u_j))\right) \tag{2.5}$$

3. **Message-Passing** : in this last case, the function ψ is a learnable message function. It takes the attribute values of nodes u_i, u_j and computes the vector of information sent from u_j to u_i. The aggregation function $\bigoplus$ can be seen as a form of message passing from $\mathcal{N}(u_i)$ towards u_i. Mathematically, it yields :

$$h_{t+1}(u_i) = \phi\left(h_t(u_i), \bigoplus_{u_j \in \mathcal{N}(u_i)} \psi_m(h_t(u_i), h_t(u_j))\right) \tag{2.6}$$

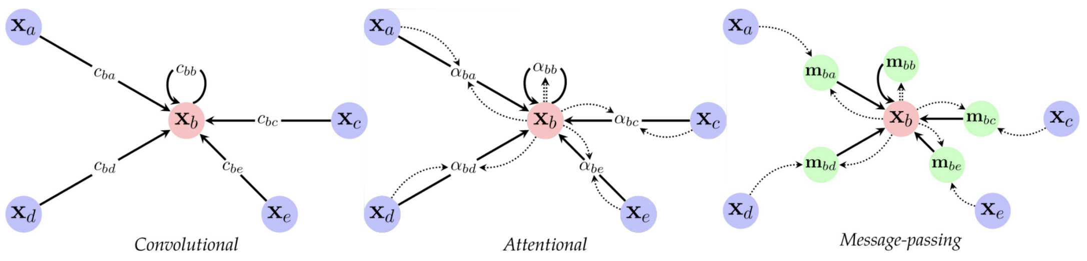


Figure 4: Visualization of dataflow from [4] for convolutional (left), attentional (middle) and message-passing (right) permutation invariant layer. The focus is on the node b and its adjacent nodes are part of $\mathcal{N}(b)$. For message-passing, the vector-based messages are defined as $m_{b,u_j} = \psi(h_t(b), h_t(u_j))$ for $u_j \in \mathcal{N}(b)$.

An interesting fact is that **convolutional** $\subseteq$ **attentional** $\subseteq$ **message-passing**. Indeed, one can for example define $a(u_i, u_j) = c_{i,j}$ to go from an attentional to a convolutional setting. For the transition between message passing to attentional, one can use $\psi_m(u_i, u_j) = a(u_i, u_j)\psi_a(u_j)$ which yields the result.

In the remainder of this work, we will only focus on graph-level tasks. This means that the GNN will output a single graph-wide embedding vector. For example, let $G(V, E)$ represent a given molecule. Then, if one wants to predict its toxicity, this task will be a graph-level problem, as it will return a single scalar value and not another graph structure. Tasks for which a GNN would return a graph structure are, for example, node-level classification tasks, where the GNN predicts a class for each node, and outputs a modified graph instance with the classes as node attributes.

Now that we have defined important graph concepts and GNNs, we will explore the graph neural network explainability field in the next section.

3 Explanations for graph neural networks

As briefly stated in the introduction of the work, the majority of explainability methods for GNNs have been oriented towards classification. In this section, we will therefore only describe the theory based on the graph-level classification tasks. The contribution of this work, which is namely the extension of several methods and metrics to the regression setting, will be described in Section 5.

In the field of explainable artificial intelligence, there is often no distinction between interpretability and explainability. While some papers use those terms as synonyms, we decide to follow the work from [25], and discriminate between the two notions. The main difference between explainability and interpretability lies in considering the model as a black box or not. More precisely, the explainability of a model refers to how well the model can be explained entirely based on input-output values. Conversely, interpretability focuses more on the mechanics of the model, and how it can interpret the decision process itself. By definition, interpretability is therefore restricted to simple models, such as linear models, which have a clear interpretation of their coefficients. Explainability often needs post-hoc analysis, via explainability methods, as it focuses on more complex and less understandable models. In this work, we will solely focus on explainability.

This theoretical section will first define what an explanation is, and how it is used for graph neural networks. Second, we will discuss the different types of explainability methods on graphs which can be used to compute explanations. Lastly, we will define metrics which help us assert the quality of the methods.

3.1 What is an explanation?

To answer this question, we will first start in Section 3.1.1 with a broad definition of an explanation, how it is defined in the graph neural network field and the definition of two focuses one might have when computing explanations for a given GNN model in Section. In Section 3.1.2, we discuss in more depth the mask weights, their mappings and links to importance scores. This section is based on the work from [2].

3.1.1 Explanation definition and focuses

Broadly speaking, an explanation is defined as the result of a process which provides structural information to the user in order to make something intelligible. The process of creating an explanation is called an explainability method. Explanations are usually important subgraphs or subsets of attributes that have a high impact on the model prediction. In the GNN setting, an explanation is a feature mask or a subgraph of the input graph, i.e. a set of features which are assigned a weight by the explainability method. This weight relates to the importance of the feature in the prediction of the model. For an input graph $G(V, E, V_{attr}, E_{attr})$ (see Section 2.1 for definition), edge and node mask are matrices of weights with domains respectively defined as

$$M_V(\hat{f}, G) \in [0,1]^N, \quad M_E(\hat{f}, G) \in [0,1]^{N \times N},$$

where $\hat{f}$ is the GNN model's prediction function. Sometimes, one also wants to explain based on the attributes of the nodes and edges, namely V_{attr}, E_{attr}, with associated masks with domains defined as

$$M_{V_{attr}}(\hat{f}, G) \in [0,1]^{N \times d}, \quad M_{E_{attr}}(\hat{f}, G) \in [0,1]^{|E| \times d'},$$

for d- and d'-dimensional node and edge attributes respectively.

From those masks on G, we define a subgraph G_S with adjacency matrix $A^S = M_E \odot A$, node features $V^S_{attr} = M_{V_{attr}} \odot V_{attr}$ and $E^S_{attr} = M_{E_{attr}} \odot E_{attr}$, where $\odot$ is the element-wise product operator. This subgraph will therefore contain all the original information from G, but the importance of the feature itself will weight the feature attributes and the adjacency matrix values. This weighting results in a new subgraph which highlights important features for the prediction of the model, which is the goal of most explainability methods for GNNs.

The masks whose values are defined in $[0, 1]$ are called *soft* masks, while *hard* masks have values defined in $\{0, 1\}$. From *hard* masks, one can further define G^t_S as a subgraph[4] of G using $M^t_E \in \{0,1\}^{N \times N}$, $M^t_{V_{attr}} \in \{0,1\}^{N \times d}$ and $M^t_{E_{attr}} \in \{0,1\}^{N \times d'}$. Those *hard* masks discard edges via the adjacency matrix and only keep selected attributes from the original embeddings of the features. Conversely to the *soft* masks, no importance weight is applied to the kept features and attributes, as their weight is equal to one. We will often call G^t_S the important subgraph, or importance-based subgraph from G, and its complement $\bar{G}^t_S$ the unimportant subgraph of G.

Following the work in [2], explaining a GNN model from its input can be done in various ways, each serving distrinct goals. First, one may want to explain why a certain decision was computed by the GNN model, for a given instance. This task is labelled with a *phenomenon* focus, where the GNN model is

[4] The exponent t stands for mask transformed, not for any time dependency.

seen as a surrogate model for the true relationship between G and the ground truth y. It implies that we ignore the GNN model parameters and its structure, and only focus on the output of the model. On the other hand, one might be interested in opening this black box and understanding the logic of the model from its outputs, which is the *model* focus. Both settings are valid complementary questions and one setting has to be selected when trying to explain a given model, as it will change the interpretation of the explainability method output. Practically, the ground-truth target y will be used for the *phenomenon* setting, and the GNN model prediction $\hat{y}$ for the *model* setting to create the input graph's explanation. The general protocol for explainability methods can be seen in Figure 5.

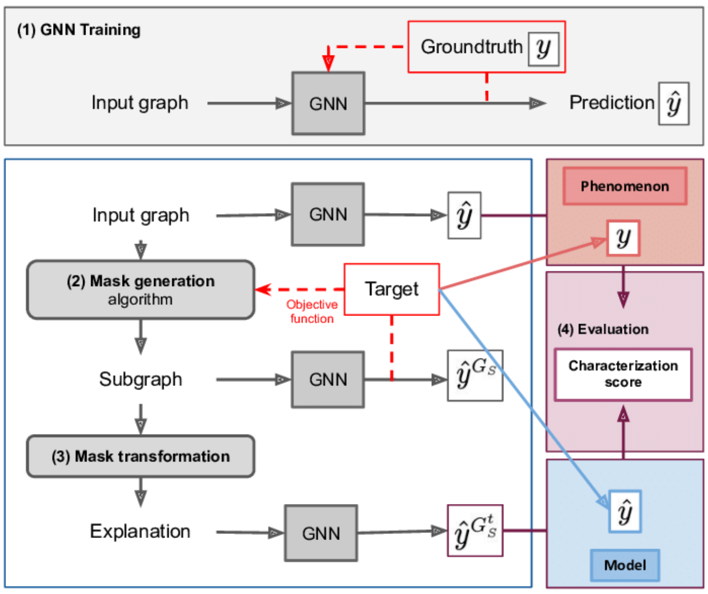


Figure 5: General protocol for GNN explanation computing from [2].

3.1.2 Discussion about masks

In this section, we discuss three methods for *soft* mask conversion, and how to interpret importance scores.

Here are described three methods from [2] for converting *soft* masks to *hard* masks.

1. **Threshold:** one can use a threshold $\tau \in [0, 1]$ representing the lowest value of explanation which is considered important. For example, if one works with edge masks, the important subgraph of a given graph $G(V, E)$ is computed as $G^t_S(V', E')$ such that

$$E' = \{\{u, v\} \in V \times V : (M_E)_{(u,v)} \geq \tau\}, \quad V' = V,$$

for a given threshold $\tau \in [0, 1]$. Analogously, one can compute the unimportant graph $\bar{G}^t_S$ as the complement graph of G^t_S (See 2.1 for the definition of complement graph).

2. **Top-k:** This method only selects the k features with highest importance value from the *soft* mask. For example, if one works with edge *soft* masks, let $\text{top}_k(E, M_E)$ be the set containing the k edges with the highest importance weights. One computes the important subgraph $G^t_S(V', E')$ such that

$$E' = \{\{u, v\} \in V \times V : \{u, v\} \in \text{top}_k(E, M_E), \quad V' = V,$$

for a given $1 \leq k \leq |E|$. The unimportant graph $\bar{G}^t_S$ is computed by retaining the features which were not in the first k most important features.

3. **Sparsity:** This methods only select a certain percentage of the total number of features, and set their importance to 1 in the computed *hard* mask, while setting the entries of the remaining features to 0, then compute G^t_S and $\bar{G}^t_S$ as previously described. One notes that this method is an "adapted" version of the *top-k* method with regards to the number of features.

However, one might be using an explainability method which only outputs a set of importance scores defined in $\mathbb{R}$ instead of masks. As previously stated, *soft* mask weights are defined on $[0,1]$ and are binary for *hard* masks. We discuss hereunder how to remap those real-defined scores to a $[0,1]$ interval, which will enable us to use the properties of masks.

Given the explainability method definition, importance scores can be interpreted in various ways. The first way is to consider that the more positive the weight, the more important the feature, while the second is to focus on absolute values of scores to model the importance of the features. In a classification setting, a highly positive importance score often means that this particular feature increases the probability of a correct class labelling, which is more related to the first way of thinking. Hence, one could map the highest element to 1, and the lowest to 0. For the other way of thinking, let M be a set of importance scores in $\mathbb{R}^k$, then one defines $\tilde{M}$ the mapped version of M to $[0,1]^k$ as

$$\tilde{M}_i = \frac{|M_i| - \min_j \text{abs}(M)_j}{\max_j \text{abs}(M)_j - \min_j \text{abs}(M)_j}, \text{ for } i = 1, ..., k,$$

where $\text{abs}(\cdot)$ is the element-wise absolute value operator.

Now that we have defined explanations, the two focuses one might have when computing explanations, and the nature of its masks, we can dive into the different types of explainability methods for graph neural networks.

3.2 Types of explainability methods on graphs

In recent years, researchers created multiple explainability methods for graph neural networks. Hereunder is an overview of the taxonomy from [25] which classes explainability methods in four different categories. This is then followed by a definition of the feature scope for explainability methods.

First, the authors define *local* explainability methods as input-dependent methods which try to explain the prediction of a GNN model for a given input graph. Conversely, they define *global* explainability methods as methods which do not use instance graphs to explain the logic of the model. Due to a low number of global explainability methods, we decided not to pursue the overview of global explainability methods.

Local explainability methods can be split into four different categories: *gradient-*, *perturbation-*, *decomposition-* and *surrogate-based* methods, as seen in Figure 6.

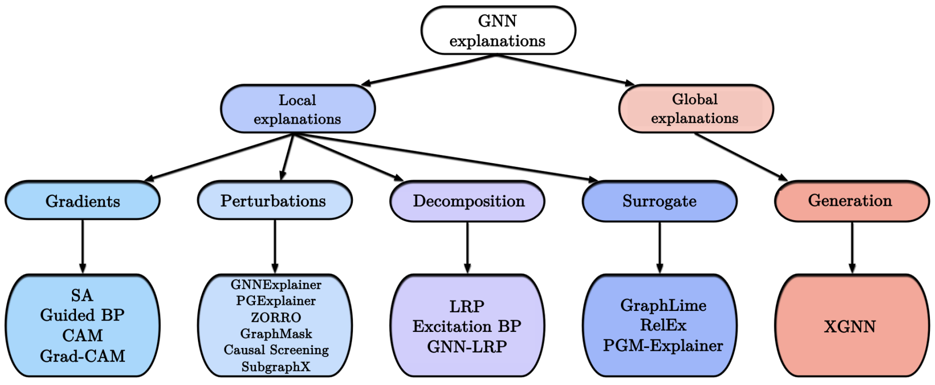


Figure 6: Taxonomy of recent explainability methods for GNNs per type of method from [25] (modified).

1. **Gradient-based**: This category of explainability methods rely on gradients from a given deep neural network to approximate the input's importance. Specifically, gradient-based methods compute gradients of the output with respect to input features by back-propagation. For classification tasks, the rule of thumb is that the larger the gradient, the more important the feature, as the probabilities will have higher variability with respect to a feature linked to a high gradient than to a low gradient.

2. **Perturbation-based**: The intuition behind this method is that if one perturbs unimportant features, then the output should remain the same as if one perturbs an important feature for the prediction, then the output might change significantly. In a graph setting, if a given set of edges is important for a prediction, then removing one might cause a big change in the prediction of the model.

3. **Decomposition-based**: This method measures the importance of the features by attributing them a given percentage of the output value using back-propagation-like techniques. This means that the sum of all the attributions of the features should be equal to the output value. In a graph setting, one can for instance distribute the total prediction score (for a classification task) to each feature of interest by back-propagation, and its related gradients.

4. **Surrogate-based**: This method relies on more simple and interpretable surrogate ML models. From a given graph and its prediction from the GNN model, the method first starts sampling a new dataset from the original graph, which can help understand the relationship between features that lead to the given prediction. In a graph setting, this can be done by removing nodes and edges or modifying their attributes. Once this sub-sampled new dataset is created, it fits several less complex and interpretable ML models to the data such as linear models or random forests, and output the computed explanations of those surrogate models.

Another important characteristic of explainability methods is which feature of the input graph they are focusing on. There exist three main categories: *node-*, *bond-* or *attribute-centric* explainers (or any combination of the them). The latter is related to nodes and edges attributes and helps one understand which dimensions of the attributes are relevant to explain a given prediction. Given its scope, an explainer will compute a mask or sub-graph of the features of interest or attributes as described in Section 3.1. In Section 4, 5 and 6, we will focus on edge-centric explainability methods for graph-level tasks.

3.3 Assessing the quality of explainability methods

To assess the quality of explainability methods, we present here different metrics which can help quantify their performance. As a reminder, we are only describing classification metrics for the moment, while the modifications to those metrics for the regression tasks can be found in Section 5.

In this section, we first describe two metrics which do not need a ground truth in their computation. This means that they only use outputs of the explainability methods and the prediction function $\hat{f}$ of the GNN model. This is useful when no ground truth is available, but limits the assessment of the metric to the comparison of computed values, as no real ground-truth importance scores are used. On the other hand, the last metric presented will need a ground truth to be used, hence permitting the comparison with real ground truth importance scores. However, this means that it is necessary to have a ground truth score for each graph on which we compute explanations, which proves to be rather restrictive in practice.

FID $\pm$ In a classification setting, the fidelity metric is defined as the difference between the accuracy obtained when removing important features and the accuracy obtained when predicting the original graph. From this definition of fidelity, one can define the sub-metrics FID $\pm$ from [25] as

$$FID+ = \frac{1}{n}\sum_{i=0}^{n} \hat{f}(G_i) - \hat{f}(\bar{G}^t_{S,i}), \quad FID- = \frac{1}{n}\sum_{i=0}^{n} \hat{f}(G_i) - \hat{f}(G^t_{S,i}), \tag{3.1}$$

where $\hat{f}$ is the prediction function of the GNN model, n is the sample size, G_i is the original graph, $G^t_{S,i}$ is the important subgraph of G_i and $\bar{G}^t_{S,i}$ its unimportant subgraph[5].

Furthermore, based on the values of $FID\pm$, authors of [2] categorize the explanation as *necessary* and/or *sufficient* :

[5] As a reminder, one can create (un)important subgraphs using *hard* masks by applying various computational methods such as the *top-k* or *threshold* methods (see Section 3.1)

- An explanation is *sufficient* if the prediction of the GNN model on the important subgraph based on the explanation is close to the prediction on the original input graph. In other words, $FID-$ has to be close to zero.
- An explanation is *necessary* if the prediction of the model is significantly different when removing the important subgraph. This means that one wants $FID+$ close to 1, as it computes the difference of prediction between the original graph and its unimportant subgraph based on the studied explainability method.

Graph explanation faithfulness (GEF) The faithfulness of a model represents how well the explanations identify input features that are important for the prediction of the model, and therefore how well they provide accurate outputs that represent the true model reasoning. The GEF is a metric which computes the unfaithfulness of a given explainer [1]. To compute this metric for a given graph G, we first need the prediction of the model $\hat{f}(G)$, as well as the prediction on the subgraph G_S^t containing its most important features, called $\hat{f}(G_S^t)$. We then have

$$GEF(\hat{f}, G, G_S^t) = 1 - \exp\big(-\mathrm{KL}(\hat{f}(G)||\hat{f}(G_S^t)\big), \tag{3.2}$$

where KL is the Kullback-Leibler divergence which quantifies the distance between two probability distributions. In this case, the higher the GEF, the worst as it computes the unfaithfulness of the explanation model.

Graph explanation accuracy (GEA) The graph explanation accuracy is a metric which compares the computed explanation with respect to ground-truth importance [1]. This metric needs *hard* masks for both the prediction and the ground truth. Using the Jaccard index [20], one gets

$$GEA(M_G^t) = \max_{M_{GT}^t \in \zeta} \frac{\mathrm{TP}(M_G^t, M_{GT}^t)}{\mathrm{TP}(M_G^t, M_{GT}^t) + \mathrm{FP}(M_G^t, M_{GT}^t) + \mathrm{FN}(M_G^t, M_{GT}^t)} \tag{3.3}$$

where $M_{\{G,GT\}}^t$ is the *hard* mask from the explanation on the input graph G, and the ground truth (denoted GT) respectively. We denote by ζ the set of ground truth masks, as several masks can explain a given graph prediction.

Now that we have presented explanations in Section 3.1, the types of explainability methods in Section 3.2, and various ways to assess them in Section 3.3, we can now present two explainability methods, still in a graph-level classification setting, in Section 4 which will then be modified in Section 5 and compared in Section 6.

4 Presentation of two explainers

In this section, we describe two explainers in the classification setting: GNNExplainer [24] in Section 4.1, and EdgeSHAPer [16] in Section 4.2. The first one can compute importance subgraphs, as well as attribute masks while the latter only focuses on edges and computes an edge *soft* mask using the Shapley value (a game theory concept, described in Section 4.2.1). GNNExplainer is therefore more general and has a broader usage than EdgeSHAPer, as it can generate masks for the node and edge attributes.

4.1 GNNExplainer

GNNExplainer was the first model-agnostic explainability method which was specifically tailored for GNNs in 2019 [24]. This explainer is part of the *Perturbation* family of explainers (see Figure 6). Given an input graph, it is able to provide the node and/or edge masks as well as the attribute mask which are considered important for the prediction of the model. The algorithm solves an optimization problem which maximizes the mutual information between a GNN's prediction and the distribution of possible subgraph structures. However, due to the complexity of generating every possible subgraph structure, a more relaxed version of the problem is practically solved by the algorithm, via a graph variational approach thoroughly described in [24]. Hereunder is a brief overview of the method.

Given a graph $G(V, E) \in \mathcal{G}$, in a graph-level classification task, the key insight on which GNNExplainer is based is that only a subgraph G_S^t of the original graph G is needed to determine the prediction of the model fully. This subgraph contains the nodes, edges and their respective attributes needed to compute the prediction. The GNN model learns a conditional distribution $P(Y|G)$ where Y is a random variable representing the class of the graph G in $\{1, ..., C\}$.

Therefore, the goal of GNNExplainer is to find a subgraph G_S^t of G which only contains the important features of G regarding the GNN model's prediction. Mathematically, one formulates the GNNExplainer optimization problem as

$$\max_{G_S^t} MI(Y, G_S^t) = H(Y) - H(Y|G_S^t), \tag{4.1}$$

where $H(Y)$ is the entropy of Y defined as

$$H(Y) = - \sum_{y=\{1,...,C\}} p(y) log(y).$$

The mutual information $MI(\cdot)$ quantifies the change of prediction probability $\hat{y} = \hat{f}(G)$ when only using G_S^t. As $H(Y)$ is constant with respect to G_S^t, one can rewrite equation 4.1 as

$$\min_{G_S^t} H(Y|G_S^t). \tag{4.2}$$

However, as stated above, this optimization problem is not tractable, as exponentially many subgraphs of G exist. One can reduce the number of submodels G_S^t is to use their adjacency matrices $A_S^t \in \{0, 1\}^{N \times N}$ and impose $0 \leq (A_S^t)_{i,j} \leq A_{i,j}$ for all $i, j \in 1, ..., N$. This will select a subset of all subgraphs of G which are sparser than the original graph G.

In practice, authors of [24] have found that one can modify the objective of equation 4.2 with a cross-entropy between the real label and the prediction of the model, which yields the optimization problem:

$$\min_{M} - \sum_{c=1}^{C} \mathbb{1}[y = c] \log P\left(Y = y|G_S^{t,(M)}\right) \tag{4.3}$$

where M is the feature mask, and $G_S^{t,(M)}$ is defined as the subgraph of G with adjacency matrix $A_S^{t,(M)} = M \odot A$. Here, one notes that we optimize on the mask M, and not on the subgraph G_S^t itself anymore. Finally, one can also add two regularizations to this problem, one on the sparsity of the subgraph and the other on entropy, which yields the final optimization problem :

$$\min_{M} - \sum_{c=1}^{C} \mathbb{1}[y = c] \log P\left(Y = y|G_S^{t,(M)}\right) + \lambda_S \operatorname{sparsity}(G_S^{t,(M)}) + \lambda_E H(G_S^{t,(M)}). \tag{4.4}$$

4.2 EdgeSHAPer

The EdgeSHAPer explainer is based on the Shapley value concept, which we define in Section 4.2.1. In Section 4.2.2, we present the EdgeSHAPer algorithm, its usage of the Shapley value in a GNN setting, and its pseudo-code.

4.2.1 Shapley value

Mathematically, one can define a cooperative game as follows: a set P of n players and a characteristic function

$$\begin{aligned} v: \quad & 2^P \to \mathbb{R} \\ & S \mapsto v(S), \end{aligned} \tag{4.5}$$

where S is a subset of P, with $v(\varnothing) = 0$. For any given coalition S, $v(S)$ gives its payoff (i.e. worth of coalition S). In this cooperative game setting, one defines the Shapley value for a player i as

$$\varphi_i(v) = \frac{1}{n} \sum_{S \subseteq P \setminus \{i\}} \binom{n-1}{|S|}^{-1} \big(v(S \cup \{i\}) - v(S)\big), \tag{4.6}$$

where $v(S \cup \{i\}) - v(S)$ can be seen at the marginal contribution of i to the payoff of the coalition and $\binom{n-1}{|S|}$ is the total number of coalitions of size $|S|$ not containing the player i. Furthermore, this definition can be interpreted as

$$\varphi_i(v) = \frac{1}{\text{Number of players}} \sum_{\text{Coalitions excluding } i} \frac{\text{Marginal contribution of } i \text{ to the coalition}}{\text{Number of coalitions excluding } i \text{ of this size}}, \tag{4.7}$$

a weighted sum of the marginal contribution of the node i over all the coalitions excluding the player i. Given equation 4.6, one can infer three main properties of the Shapley value (without proof):

1. **Efficiency**: one has that $\sum_{i=1}^{|N|} \varphi_i(v) = v(N)$, meaning that the sum of all marginal contributions equals the total payoff of the full graph.
2. **Symmetry**: if $v(S \cup \{i\}) = v(S \cup \{j\})$, then $\varphi_i(v) = \varphi_j(v)$, also called the equal treatment of equals.
3. **Linearity**: $\varphi_i(\alpha v + \beta w) = \alpha\varphi_i(v) + \beta\varphi_i(w)$ for any player i in N.

Any valid choice of $v(\cdot)$ will therefore need to satisfy those three properties (as well as $v(\varnothing) = 0$). As Shapley values are defined in $\mathbb{R}$, it means that each player will contribute positively or negatively to the payoff, and the sum of their Shapley value will give the total payoff. One drawback of Shapley values is that the dependence of one or more players together is omitted. For example, this might become problematic if two players excel as a pair, but perform poorly individually. By definition, the Shapley value will not catch this relation between the two players and will give a low value to each of them, even though they can be the best performers if both are in the coalition.

4.2.2 EdgeSHAPer definition and algorithm

EdgeSHAPer [16] is a generally applicable explainability method based on the Shapley value to assess edge importance for the prediction of GNNs. The Shapley value's game theory principle has been adapted to graphs by using the following analogies: edges correspond to players and work collectively as a coalition, and the graph is made from those edges and their incident nodes. The coalition's payoff represents the probability of its correct class label prediction. Lastly, the individual contribution of each edge towards the payoff is its Shapley value. With those analogies, one can rewrite equation 4.6 as

$$\varphi_i(v) = \frac{1}{|E|} \sum_{S \subseteq E \setminus \{i\}} \binom{|E|-1}{|S|}^{-1} \big(v(S \cup \{i\}) - v(S)\big), \tag{4.8}$$

where E is the edge set, S indicates all the possible subsets of edges excluding i.

Given the definition of the Shapley value, which sums over all coalitions S, one finds that its computation is not tractable, and will therefore have to be approximated[6]. To approximate the Shapley value of each edge, authors of [16] define a Monte-Carlo strategy that generates random graphs from an input graph $G(V, E)$. This random graph $Z(V_Z, E_Z)$ is defined with $V_Z = V$, and

$$E_Z = \{e \in E : Pr(e \in E_Z) = \text{Ber}(p)\} \text{ with } p = \rho(G),$$

where $\rho(G)$ is the edge density of G. This means that an existing edge in E will exist in E_Z following $\text{Ber}(p)$, where $p = \rho(G)$. This type of random graph generation where each edge has a fixed probability of being present or absent is called an Erdos–Rényi model. In this model, all graphs with the same number of edges, are equally likely.

Algorithm 1 shows the EdgeSHAPer pseudo-code, which takes as arguments $G(V, E)$, the input graph, j the edge index for which we compute the explanation, p the probability for edges to be in Z, M the number of Monte-Carlo simulations and $\hat{f}$ the prediction function of the GNN model. One finds that there is a splitting point at index j such that for E_{+j}, the jth edge comes from E while for E_{-j}, it originates from E_Z. This splitting point helps computing the marginal contribution of edge j to the output as it is the only different element between E_{+j} and E_{-j}. Convergence-wise, the error decreases in $\mathcal{O}(M^{-\frac{1}{2}})$ which is the typical rate of convergence for Monte-Carlo methods.

[6] One notes that the same tractability problem arose for the GNNExplainer when one had to compute all subgraphs of the input graph, even though here we are selecting subsets of the edge set.

Algorithm 1: EdgeSHAPer with Monte-Carlo sampling

```
Data: $G(V, E), j, p, M, \hat{f}$
Result: returns approximated Shapley value $\varphi_j(v)$ for a given edge $j$.
$cumulative_{\phi_j}(G) \leftarrow 0$;
foreach $m \in \{0, ..., M-1\}$ do
    $V_z \leftarrow V$ // creating the node set for $Z$
    define $E_z = \{z \in E : \Pr(z \in E_z) = p\}$ // creating the edge set for $Z$
    define graph $Z(V_z, E_z)$ // generating random graph $Z$
    $\pi \leftarrow \text{perm}(|E|)$ // creates a permutation of $\{0, ..., |E|-1\}$
    $j^\pi \leftarrow \text{index}(j, \pi)$ // gets the permuted index $j$
    $E^\pi \leftarrow \text{sort}(E, \pi)$ // sorts the edge set of $G$ following $\pi$
    $E_z^\pi \leftarrow \text{sort}(E_z, \pi)$ // sorts the edge set of $Z$ following $\pi$
    $E_{+j} \leftarrow (e_0, ..., e_{j^\pi}, z_{j^\pi+1}, ..., z_{|E|-1})$ // concatenates the two edge sets with a splitting
        point at $j^\pi$
    $E_{-j} \leftarrow (e_0, ..., e_{j^\pi-1}, z_{j^\pi}, ..., z_{|E|-1})$ // concatenates the two edge sets with a splitting
        point at $j^\pi - 1$
    $\phi_j^m(G) \leftarrow \hat{f}(E_{+j}) - \hat{f}(E_{-j})$ // subtracts both predictions from the given edge sets
    $cumulative_{\phi_j}(G) \leftarrow cumulative_{\phi_j}(G) + \phi_j^m(G)$
end
$\varphi_j(G) \leftarrow cumulative_{\phi_j}(G)/M$
```

In Section 3 and 4, we have described important explainability concepts, and two explainability methods for GNNs, for classification tasks. In the following section, we discuss the necessary modifications and difficulties which occur to transition to the regression setting of the concepts previously described.

5 Specificities and modifications for regression

5.1 Explanations

Switching to regression tasks, the definition of an explanation and its focuses (Section 3.1.1) stay the same, but there are changes in the way of understanding positive and negative importance scores and metrics to assess method performances.

5.1.1 Understanding of positive and negative importance scores

We stated in Section 3.1.2 that the feature with the highest importance score will contribute the most to the probability of a correct class labelling, while the lowest will be the feature which contributes the least. For regression tasks, the importance scores are seen as positive or negative contributions to the numerical prediction of the model. This yields that the magnitude of the importance score is more important than its sign. Hence, large scores in absolute value will be given to features which impact positively or negatively the final numerical prediction.

Remapping the importance scores to weights in $[0, 1]$ to obtain *soft* masks will therefore be made by taking the absolute value of the scores, and then remapping them to $[0, 1]$. As a reminder, let $M \in \mathbb{R}^k$ the importance scores, then we define $\tilde{M}$ the mapped version of M to $[0, 1]^k$ as

$$\tilde{M}_i = \frac{|M_i| - \min_j \text{abs}(M)_j}{\max_j \text{abs}(M)_j - \min_j \text{abs}(M)_j}, \text{ for } i = 1, ..., k.$$

Regarding the types of explainability methods in Section 3.2, those are task-invariant, and do not need to be modified for the regression setting. However, we need to update the Fidelity metric in Section 3.3, to adapt it to the regression setting.

5.1.2 Fidelity metric $FID\pm$

First, we have to define a regression setting fidelity metric, as the accuracy metric is no more available. Therefore, we decided to replace the difference in accuracy with the difference in the model predictions. Furthermore, if one assumes that an over- or undershoot of the prediction results in the same error, it comes naturally to consider the absolute value of the difference, which yields the new $|FID|\pm$ metrics defined as

$$|FID|+ = \frac{1}{n}\sum_{i=0}^{n} \frac{|\hat{f}(G_i) - \hat{f}(\bar{G}^t_{S,i})|}{|\hat{f}(G_i)|}, \quad |FID|- = \frac{1}{n}\sum_{i=0}^{n} \frac{|\hat{f}(G_i) - \hat{f}(G^t_{S,i})|}{|\hat{f}(G_i)|}, \tag{5.1}$$

where $\{G_i\}_{i=1,\dots,n}$ is a set of graphs, G^t_S and $\bar{G}^t_S$ are respectively the important and unimportant subgraph of G, and $\hat{f}$ is the model prediction function. We decided to divide the difference by the absolute value original prediction, as it helps us to express it in percentages and sum efficiently over each graph in $\{G_i\}_{i=1,\dots,n}$. For $|FID|\pm$, a good GNN explanation would have a high $|FID|+$ and low $|FID|-$.

Regarding the *sufficient* character of an explanation, we still want $|FID|-$ close to zero. However, now that the metrics are not bounded in $[0, 1]$, the *necessary* character of an explanation will be described by a high $|FID|+$.

For the two other presented metrics, the procedure does not change for the regression setting, as we still rely on the *top-k* method for mask creation described in Section 3.1.2. Now that we presented the necessary modifications for Section 3 for the regression setting, we can explore the modifications for the explainers presented in Section 4.

5.2 Explainers

In this section, we will cover the modifications needed to use the two presented explainers for a regression task. Regarding the GNNExplainer, we only need to change the loss from equation 4.4 to a mean squared error loss, while we keep both regularization terms for sparsity and entropy[7]. We will now focus on the modifications to the EdgeSHAPer algorithm, and then present another algorithm which is a variant of the modified EdgeSHAPer.

5.2.1 EdgeSHAPer modifications for regression

Conceptually, we had to modify the usage of the Shapley value for the regression task. More precisely, we changed the payoff v from the probability of a correct class label prediction to the graph-level scalar prediction of the model, such that we define the regression-based scoring function $\tilde{v}$ as

$$\begin{aligned} \tilde{v}: \quad & 2^{|E|} \times \mathcal{G} \to \mathbb{R} \\ & (S, G) \mapsto \hat{f}(G(V, S, V_{attr}, S_{attr})), \end{aligned} \tag{5.2}$$

where S is a subset of E, $\hat{f}$ is the prediction function from the GNN model for an input graph $G(V, E)$. This means that the Shapley value now describes the marginal difference of the prediction with or without a given edge. Thanks to the efficiency property of the Shapley value, one gets with that scoring function that the sum of the edge Shapley values should be equal to the model's prediction on the original graph.

Practically, we modified the computation of the Shapley value with the usage of the model's prediction on $E_{\pm j}$ (see Alg. 1 for notation) which were using log-odds and softmax activation layer for classification purposes (get the probabilities between 0 and 1) to the difference of the model's prediction.

Another broader modification from the initial EdgeSHAPer algorithm was to allow the prediction functions to take edge attributes as parameters. Originally, only the nodes, edges and node attributes were inputs of the prediction function of the GNN model.

For this modified version of the EdgeSHAPer algorithm, we kept the original random sampling method to create random graphs Z and compute edge marginal contributions. However, we tried to find another

[7] See the PyG implementation of the GNNExplainer optimization problem here: `https://github.com/pyg-team/pytorch_geometric/blob/master/torch_geometric/explain/algorithm/gnn_explainer.py`

random sampling method which would better replicate the combinatorial distribution of subsets from the edge set. This new algorithm will be described in Section 5.2.2. Please note that starting from the next section, we will use the term EdgeSHAPer to refer to the regression-setting modified EdgeSHAPer algorithm for clarity purposes.

5.2.2 regSHAPer

Based on EdgeSHAPer described earlier, one observes the usage of a permutation-based method, with a uniformly sampled random graph Z, to compute the marginal contribution of a given edge. The motivation for this new method is to change the subgraph generation by randomly sampling graphs with a different approach. For a given graph $G(V, E)$, we compute e_i's importance by uniform sampling from the combinatorial-based distribution of subgraphs $\tilde{G}(V', E')$, with $E' \subseteq E \setminus e_i$, and then compare the predictions of $\tilde{G}(V', E')$ to $\tilde{G}(V', E' \cup \{e_i\})$. We further discuss this sampling method hereunder.

Let the edges of a given graph $G(V, E, V_{attr}, E_{attr})$ be the players and $v : 2^{|E|} \to \mathbb{R}$ be the payoff function of a given coalition. One can start reasoning about the algorithm to compute the Shapley value. If we have to sum over all partitions of E excluding one edge, this would need

$$T := \sum_{k=0}^{|E|-1} \binom{|E|-1}{k} = 2^{|E|-1}$$

iterations, which is not tractable. Therefore, one has to implement a "Monte-Carlo" type of algorithm by randomly creating a few subsets of edges not containing the studied edge. To do so, one can compute the number of subsets of size

$$k := \binom{|E|-1}{j} \text{ for each } j \in \{0, ..., |E|-1\},$$

then dividing it by T which will give us the approximate proportion of the subsets of E of size k, called P_k. Then, let M the number of simulated subsets of E that can be generated, one gets round(MP_k) subsets in the class C_k, the set of generated subsets of size k. Finally, we can express our estimated Shapley value as

$$\hat{\varphi}_i(v) = \frac{1}{|E|} \sum_{k=0}^{|E|-1} \sum_{S \in C_k} \frac{(\tilde{v}(S \cup \{e_i\}) - \tilde{v}(S))}{|C_k|}. \tag{5.3}$$

This yields the algorithm 2, which is based on the EdgeSHAPer algorithm architecture, as seen in algorithm 1, with the modified sampling and scoring function.

6 Experiments

In this section, we apply concepts and explainers presented in Section 4 and modified in Section 5. More precisely, we compare three explainers (GNNExplainer [24], EdgeSHAPer [16] and regSHAPer on the PCQM4Mv2 dataset [17] which is a quantum chemistry dataset and analyze the results. We decided to focus on a chemistry dataset for two main reasons. First, one can use graphs to model molecules as molecular graphs are the primary data structure for conveying molecular structure, chemistry structure properties, and atom or bondwise interactions as seen in Figure 7. Second, the dataset used in this project had relevant ground truth explanations [9] which could be helpful for assessing the performance of an explainer using previously described metrics.

Algorithm 2: regSHAPer algorithm

```
Data: $G(V, E, V_{attr}, E_{attr}), i, M, \hat{f}$
Result: returns approximated Shapley value $\varphi_i(\tilde{v})$ for a given edge $i$.
$\varphi_j(G) \leftarrow 0$
$T \leftarrow \sum_{k=0}^{|E|-1} \binom{|E|-1}{k}$ // Total number of subsets of size $k$ of $E$
$P \leftarrow \{\binom{|E|-1}{k}/T : k = 0, ..., |E|-1\}$ // Defining the $P_k$'s
$C \leftarrow \text{round}(PM)$ // Defining the $C_k$'s
for $k \in \{0, ..., |E|-1\}$ do
    // looping over every class $C_k$
    $S := \{S_i : i = 1, ..., |C_k|\}$ where $S_i$ are samples of size $k$, uniformly drawn from $E \setminus \{e_i\}$
    $cum_{\varphi_i(\tilde{v})} \leftarrow 0$
    foreach $S_j \in S$ do
        $cum_{\varphi_i(\tilde{v})} \leftarrow cum_{\varphi_i(\tilde{v})} + (\tilde{v}(S_j \cup \{e_j\}) - \tilde{v}(S_j))$, where $\tilde{v}(S_j) = \hat{f}(N, S_j, N_{attr}, S_{attr,j})$
    end
    $\varphi_j(G) \leftarrow \varphi_j(G) + cum_{\varphi_i(\tilde{v})}/|C_k|$
end
$\varphi_j(G) \leftarrow \varphi_j(G)/|E|$
```

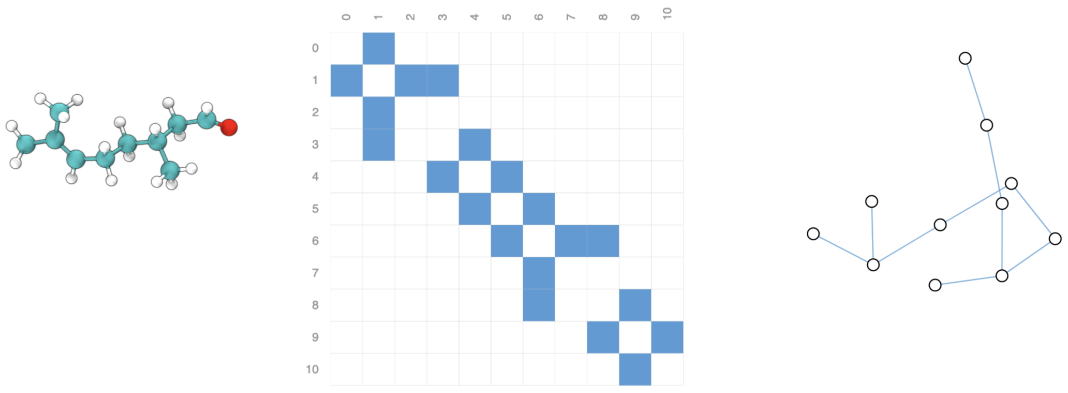


Figure 7: Comparison of molecule representation from [19]. From left to right, there is the 3D molecular graph, then the adjacency matrix, and lastly the converted molecule to a graph object.

6.1 Dataset description

The HOMO-LUMO gap is a quantum chemistry property which represents the energy difference between the Highest and Lowest Occupied Molecular Orbital for the ground state of a molecule. For a given molecule, one can empirically compute its HOMO-LUMO Gap, which has been done by the PubChemQC project [17], and made publicly available in the PCQM4Mv2 dataset.

This dataset contains molecules and their HOMO-LUMO gap values. Regarding the format of the molecules, it is given as $3D$-structures which can be efficiently interpreted by computers. Practically, we used the Pytorch Geometric[8] [6] (PyG) compatible version of the dataset, where a given molecule is defined by a set of nodes (atoms), edges (chemical bonds) and various node and edge attributes. Additionally, this format of the molecule is compatible with the chosen GNN model, which is written based on the PyG library.

For the experiment, we wished to use the ground truth dataset on HOMO-LUMO gaps from [9]. Unfortunately, there was no correspondence between the edge indices from the PyG graphs and the ones from the ground truth dataset. Due to limited time, we chose not to pursue the matching and will therefore

[8] Pytorch Geometric is a library built upon Pytorch made to write, train and use graph neural networks.

not be able to test our methods using ground truth-based metrics such as *GEA*. For more information regarding the explication of the creation of this ground truth dataset, please see Appendix A.

6.2 Model description

The graph neural network model from [10] used in this project is a graph convolutional network (GCN). It is made out of several hidden layers which share the same permutation-invariant layer defined as

$$h_{t+1}(x_i) = \mathbf{\Theta}_t^\top \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{e_{j,i}}{\sqrt{\hat{d}_i \hat{d}_j}} h_t(x_j), \tag{6.1}$$

where $\hat{d}_i = 1 + \sum_{j \in \mathcal{N}(i)} e_{j,i}$, and $e_{j,i}$ is the edge weight from $\{j, i\}$. The vector $\mathbf{\Theta}_t \in \mathbb{R}^d$ (d is the dimension of node attributes) is a vector of learnable weights for the sum of embeddings from the neighbourhood of a node for the t^{th} iteration.

We chose this GCN model from [10] as it performs well for the HOMO-LUMO gap problem (see the leaderboard), it is also simple to understand by only using GCNs (only 2×10^6 parameters) and it is computationally efficient (less than one second per prediction). Regarding its metrics, it has a *MAE* for the validation set of 0.14 eV and 0.15 eV for the test set. Those are considered quite low, knowing that the chemical accuracy when empirically computing the HOMO-LUMO gap is 0.04 eV [10], and the mean HOMO-LUMO gap from the dataset is 3.01eV.

6.3 Validation and practical tests of the regSHAPer algorithm

After running some tests, we noted several interesting observations regarding this algorithm, which help us validate the explainability method and safely use it in the comparison with the GNNExplainer and EdgeSHAPer algorithms.

1. First, when one creates the subgraph from $G(V, E)$ made from a given subset of edges, we obtained infinite values for Shapley values with the chosen GNN model. This was due to keeping nodes which are no longer connected to any given edge (their degree was zero). It caused problems during the pooling phase when computing the predictions, as we are using *sum*-pooling at each layer. A workaround has been found: creating two batches for each prediction. The first batch would contain all nodes with degrees bigger or equal than one, and the second batch would contain nodes which are disconnected. We then run the model on those two batches and only keep the prediction of the first one, while discarding the second prediction. This is equivalent to dropping isolated nodes, but this batch method avoids modifying the graph structure and node attributes matrix, which is computationally more efficient.

2. Given a graph small enough (e.g. 7 edges), we were able to compute all possible combinations of edges, hence creating the 64 possible subgraphs. Thanks to this graph, we were able to check that the efficiency property of the Shapley value was satisfied, which is a good sign for the choice of the scoring function $\tilde{v}$. Namely, when summing all Shapley values for the seven edges, we obtained the same value as the prediction of the model for the entire graph containing the seven edges, up to the fourth decimal point.

3. Another small workaround that had to be done is when tackling the $S = \varnothing$, as the model would not be able to predict a HOMO-LUMO gap value for an empty graph. Therefore, when we want to compute $v(\varnothing)$, we simply set it to zero, which is another desired property.

6.4 Results

The main parts of the experiments were the training of the model, running the explainability methods on a set of molecules, and lastly the comparison of the methods. This section will tackle the last part, regarding the comparison of the explainability methods, while the experiment methodology, and more generally all practical information regarding the experiments can be found in Appendix C. All the code for reproducibility can be found on `https://github.com/evandieren/GNN_Explainability`. Throughout this section, the explainability methods were all explainaing the same model described in Section 6.2.

6.4.1 Prediction vs. *top-k* prediction, fidelity and faithfulness

First, we start by comparing the three explainability methods regarding the closeness of their prediction to their *top-k* prediction. By doing such an analysis, we try to check whether the prediction on the subgraph made of the *top-k* most important edges according to each method is close to the original prediction. If they are close, it means that the explainability method was avle to correctly identify important features from the original graph. The best case would be to have all points on the diagonal line, which mean that the *top-k* prediction would be equal to the original prediction. The worst case would be to have no-trend at all and generally points far away from the diagonal line. One can link this analysis to the *sufficient* character of the method, as it tells us how G_S^t, the important subgraph of features, differs from the original prediction.

In this section, we decided to work on the first 5000 molecules from the PCQM4Mv2 dataset[9] and to set $k = 5$ (plots on other values of k can be seen in Appendix D). This value of k has been chosen to get small enough subgraphs, knowing that the average number of bonds for the 5000 chosen molecules is 13, but ranges from 5 to 20.

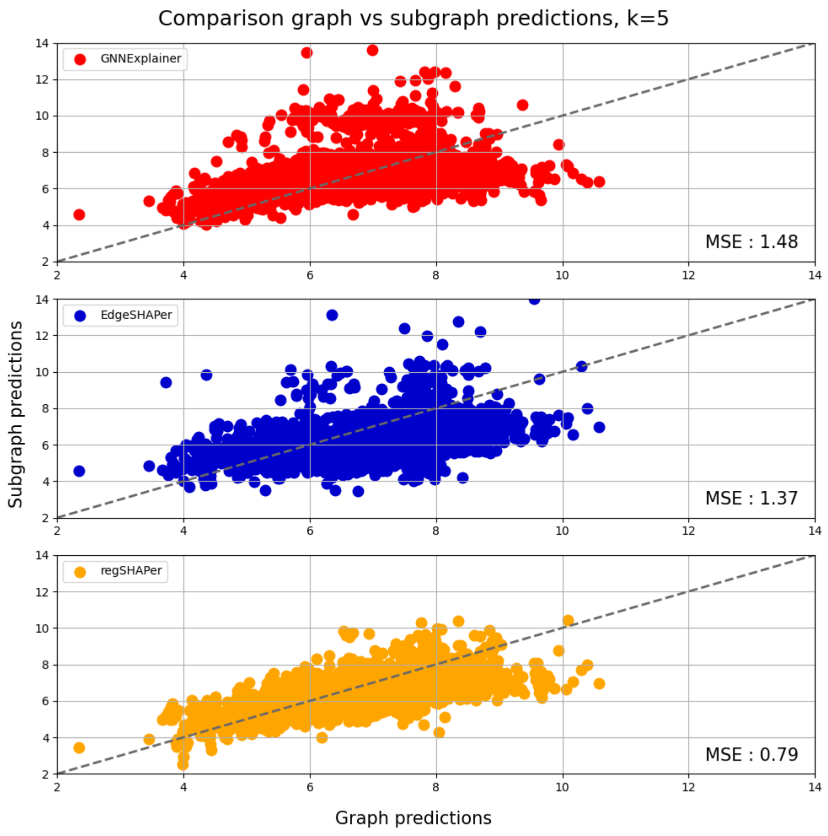


Figure 8: Comparison of GNN model's prediction on *Top-5* subgraph explanations (y-axis) vs on the original graph (x-axis). The diagonal line represents the explainability method optimality, where both predictions are equal. The main message is that the regSHAPer is closer to the optimal line (smaller MSE), while the two other methods are more widespread, and have an apparent overshoot from *Top-5* subgraph predictions. Prediction on 5000 molecules from the training dataset. All the methods explain the same model.

From Figure 8, we first observe that the subgraph prediction often overshoots the original graph prediction for the GNNExplainer and EdgeSHAPer, as lots of dots are far above the diagonal line. Generally speaking, we observe that the regSHAPer method fits better the optimal diagonal line than the two other methods. This is confirmed when computing the Mean Squared Error (MSE) between the points and the

[9] The 5000 molecules have also been used for the training of the model.

diagonal optimal line. We observe that GNNExplainer has the highest MSE, followed by EdgeSHAPer and then regSHAPer, which coincides with the points spreading on the three plots. This behaviour remains constant with respect to variation of k, as seen in Appendix D.

One can also be interested in the metrics described in Section 3.3 and modified in Section 5 to compare the three explainability methods quantitatively. Table 1 summarizes all the computed metrics for each explainability method, and are discussed hereunder.

1. From the $|FID|-$ column, we can link it to the *sufficient* character[10] of explainability methods and observe that the regSHAPer has a lower value than the two other explainability methods. This means that the regSHAPer method provides important subgraphs G^t_S which are of slightly better quality, as its prediction will only be 10% different than the original graph prediction on average, compared to 13% for the two other methods. However, one notes that the standard deviation magnitude is quite high, especially for the EdgeSHAPer and the GNNExplainer which are twice more unstable compared to the regSHAPer.

2. From the $|FID|+$ column, we can link it to the *necessary* character[11] of explainability methods and observe that both GNNExplainer and regSHAPer have similar slightly higher values than the EdgeSHAPer algorithm on average. From those values, we conclude that the predictions on unimportant subgraphs $\bar{G}^t_S$ are approximately 14% different compared to the prediction on the original graph. Once again, one notes that those metrics have a high standard deviation, and no explainability method emerges as more stable.

3. Lastly, another link can be made to faithfulness[12], which can be assessed with the GEF metric defined in Section 3.3. The GEF column in the table shows us that the regSHAPer is significantly better compared to the two other methods when comparing their faithfulness, as its GEF value is 17 and 33% less than EdgeSHAPer and GNNExplainer respectively. By construction of GEF, this deterministic metric does not have any variance.

Table 1: $|FID|\pm$ and GEF values for the three explainability methods, 5000 molecules.

	$\lvert FID\rvert - \downarrow$	$\lvert FID\rvert + \uparrow$	$GEF \downarrow$
GNNExplainer	0.131 $\pm$ 0.13	**0.146** $\pm$ 0.10	0.402
EdgeSHAPer	0.131 $\pm$ 0.1	0.136 $\pm$ 0.09	0.362
regSHAPer	**0.100** $\pm$ 0.07	0.143 $\pm$ 0.09	**0.266**

6.4.2 Threshold analysis

For GNN explainability methods, it is preferable to find distinguishable importance sub-graphs. Therefore, one would wish the explainability method to give a clear separation between the importance values, to be certain about each feature's importance ranking. As a toy example, if one has two importance weights defined as

$$I_1 = [0.2, 0.30, 0.05, 0.45], \quad I_2 = [0.24, 0.23, 0.26, 0.27],$$

one would prefer I_1 over I_2 as it has more clear separations than the latter, which rankings are much more subject to noise, due to its values closeness.

Furthermore, it is also important for an explainability method to output sparse subgraphs, as it will be easier for users to identify relevant patterns across graphs. Hence, studying the magnitude of importance weights is interesting, as one can potentially observe whether the explainability methods correctly find a sparse subgraph or if it gives importance to lots of features in a given graph.

In our case, we assess this characteristic using a threshold value $\tau \in [0, 1]$ and report the percentage of edges whose importance is above the threshold when varying it from 0 to 1. To make the explainability

[10] One recalls that an explanation is considered *sufficient* if the prediction of the GNN model on the subgraph containing important features based on the explanation is close to the prediction on the original input graph.
[11] An explanation is considered *necessary* if the prediction of the GNN model is significantly different compared to the prediction on the original graph when one removes the important subgraph created by the explanation.
[12] As a reminder, the faithfulness of a model is that the explanations should identify input features that are important for the model's prediction

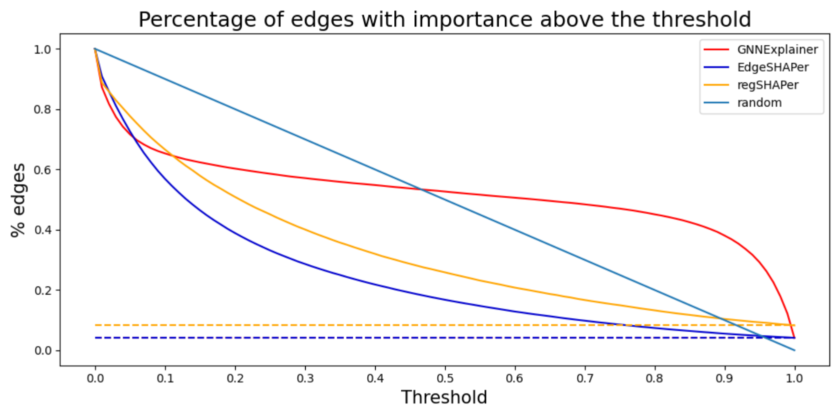


Figure 9: Percentage of edges (y-axis) for which the *soft* mask provides an explanation which is bigger than the threshold (x-axis). The diagonal straight line represents a random uniform explainer which would assign uniform random weights in $[0, 1]$ to the edges. The dashed lines mark the average inverse of the number of edges for the 5000 molecules used. The main result is that both EdgeSHAPer and regSHAPer have similar results when compared to the GNNExplainer. This means that generally, the Shapley-based explainers generate sparser subgraphs after thresholding.

methods comparable, we took the absolute value of any negative importance (see Section 5 for the negative importance interpretation in a regression setting), then linearly remapped the values in the $[0, 1]$ space.

For a subgraph to be sparse, one wants the curves from Figure 9 to decrease at the beginning steeply, because this means that lots of features would not have a significant importance, as they would be eliminated with a small importance value. Here, we observe similar behaviour at the very beginning, with the three curves from the explainability methods that decrease steeply. Then, at around $\tau = 0.05$, we observe a divergence from the GNNExplainer algorithm, which will reduce its steepness until $\tau = 0.5$. This means that the GNNExplainer algorithm has difficulties finding irrelevant features and keeps approximately 70% of its edges up to $\tau = 0.5$. Regarding the two other explainability methods, we observe similar shapes and results, with the EdgeSHAPer curve being slightly steeper. One notes that the dashed line is the expected percentage of edges having the lowest importance (namely, the mean of the inverse of the number of nodes, over every molecule) and is used to verify that the percentage reached by the explainers is correct.

This sparsity can also be seen in Figure 10, where the edge colour represents the importance weight of the *soft* mask computed by each explainability method.

To conclude this result section, we observed that the three explainability methods did not perform equivalently for the different tests which have been made. We highlight that regSHAPer almost always topped the two other methods, except the $|FID|+$ metric, where it was 0.3% behind the GNNExplainer on average. However, this comparison is not exhaustive and has several limitations. First, we observed a high variance for the $|FID|\pm$ metrics, which make those two metrics conclusions less trustworthy. Second, we ran the comparison on only 5000 molecules which were taken from the training set. Third, the metrics were tested on a single GNN model. Lastly, we did not have ground truth importance scores to compare with the computed explanations, hence no real-world comparison has been done.

7 Discussion

The main contribution of this work is the advancement in adapting existing GNN explainability methods and metrics from classification to regression setting. These modifications encompass both conceptual and practical aspects. Conceptually, we emphasize the shift in interpreting importance scores from explainability methods. In the regression setting, we chose a different perspective on feature importance compared to the classification setting. Instead of interpreting highly negative score features as unimpor-

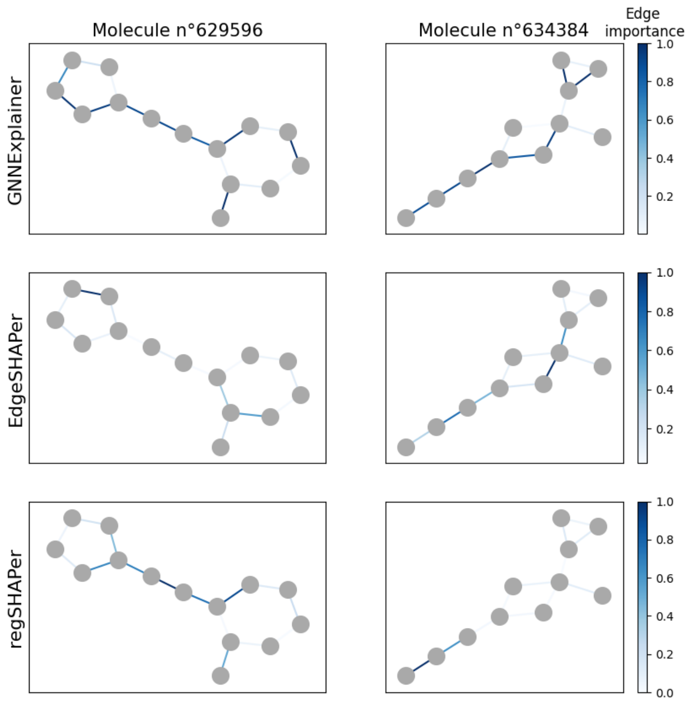


Figure 10: Visualization of explanations for GNNExplainer, EdgeSHAPer and regSHAPer (top to bottom) for two molecules (left to right). All explanations have been remapped to the interval $[0, 1]$ with 1 being the highest importance weight of the explanation, and 0 the lowest. We observe different edge importance *soft* masks for each explainability method on the two molecules. On the left, one notes that the GNNExplainer considers a higher number of edges as important compared to the two other explainers. We say that it has a lower sparsity. Additionally, the most important edge for the EdgeSHAPer is overlooked by both the regSHAPer and GNNExplainer. On the right, we observe similar results regarding the sparsity of the explanation. However, all methods for the second molecule consider the bottom left bonds of the molecule important.

tant, we focused on the absolute values of scores as a more representative indicator of feature importance. Regarding practical modification, we have presented adjustments to the EdgeSHAPer algorithm to extend it with edge-centric explanations for regression. Additionally, we decided to modify its underlying random graph sampling mechanism and introduced one which was more aligned with the combinatorial distribution of subsets of the edge set. This modified sampling, combined with the existing EdgeSHAPer architecture, resulted in the regSHAPer. This explainer demonstrated promising performances compared to the two other methods in the preliminary results showed in Section 6.4. However, it is important to note that there were some limitations to this comparison. First, the three methods were only tested on a single model and a single dataset. Second, the absence of ground truth dataset prevented a comparison between computed explanations and empirical importance scores.

Promising future directions for this work include the following: first, one could evaluate the methods against ground truth data, hence enabling a more rigorous and extensive comparison between computed explanations and empirical importance scores. Second, integrating the computed explanations for the regression setting into decision-making processes could be explored to assess their impact on decision outcomes. Lastly, a more broad and theoretical direction would be to investigate the necessity of connectivity in important subgraphs, and quantify how valuable disconnected importance subgraphs are to GNN explanations.

These three future steps aim to futher advance the field of GNN explainability for regression, and improve transparency of GNN decisions for high-risk domains such as drug-discovery, chemistry and medical imaging.

References

[1] Chirag Agarwal, Owen Queen, Himabindu Lakkaraju, and Marinka Zitnik. Evaluating explainability for graph neural networks. *Scientific Data*, 10(1):144, 2023. [Cited on pages 1 and 10.]

[2] Kenza Amara, Rex Ying, Zitao Zhang, Zhihao Han, Yinan Shan, Ulrik Brandes, Sebastian Schemm, and Ce Zhang. Graphframex: Towards systematic evaluation of explainability methods for graph neural networks. *arXiv preprint arXiv:2206.09677*, 2022. [Cited on pages 6, 7, and 9.]

[3] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. [Cited on page 1.]

[4] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021. [Cited on pages 3, 4, and 5.]

[5] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *nature*, 542(7639):115–118, 2017. [Cited on page 1.]

[6] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019. [Cited on pages 1 and 16.]

[7] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018. [Cited on page 1.]

[8] Martin Hanik, Mehmet Arif Demirtaş, Mohammed Amine Gharsallaoui, and Islem Rekik. Predicting cognitive scores with graph neural networks through sample selection learning. *Brain imaging and behavior*, pages 1–16, 2021. [Cited on page 1.]

[9] Eugen Hruska, Liang Zhao, and Fang Liu. Ground truth explanation dataset for chemical property prediction on molecular graphs. *ChemRxiv preprint*, 2022. doi: 10.26434/chemrxiv-2022-96slq-v2. [Cited on pages ii, 15, 16, and 24.]

[10] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021. [Cited on pages 1, 17, and 25.]

[11] Tobias Skovgaard Jepsen, Christian S Jensen, and Thomas Dyhre Nielsen. Graph convolutional networks for road networks. In *Proceedings of the 27th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 460–463, 2019. [Cited on page 1.]

[12] Dejun Jiang, Zhenxing Wu, Chang-Yu Hsieh, Guangyong Chen, Ben Liao, Zhe Wang, Chao Shen, Dongsheng Cao, Jian Wu, and Tingjun Hou. Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models. *Journal of cheminformatics*, 13(1):1–23, 2021. [Cited on page 1.]

[13] Mengyuan Lee, Guanding Yu, Huaiyu Dai, and Geoffrey Ye Li. Graph neural networks meet wireless communications: Motivation, applications, and future directions. *IEEE Wireless Communications*, 29(5):12–19, 2022. [Cited on page 1.]

[14] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017. [Cited on page 1.]

[15] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. *Advances in neural information processing systems*, 33:19620–19631, 2020. [Cited on page 1.]

[16] Andrea Mastropietro, Giuseppe Pasculli, Christian Feldmann, Raquel Rodríguez-Pérez, and Jürgen Bajorath. Edgeshaper: Bond-centric shapley value-based explanation method for graph neural networks. *Iscience*, 25(10):105043, 2022. [Cited on pages 1, 10, 12, and 15.]

[17] Maho Nakata and Tomomi Shimazaki. Pubchemqc project: a large-scale first-principles electronic structure database for data-driven chemistry. *Journal of chemical information and modeling*, 57(6): 1300–1308, 2017. [Cited on pages 15 and 16.]

[18] Patrick Reiser, Marlen Neubert, André Eberhard, Luca Torresi, Chen Zhou, Chen Shao, Houssam Metni, Clint van Hoesel, Henrik Schopmans, Timo Sommer, et al. Graph neural networks for materials science and chemistry. *Communications Materials*, 3(1):93, 2022. [Cited on page 1.]

[19] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. doi: 10.23915/distill.00033. https://distill.pub/2021/gnn-intro. [Cited on pages 4 and 16.]

[20] Abdel Aziz Taha and Allan Hanbury. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. *BMC medical imaging*, 15(1):1–28, 2015. [Cited on page 10.]

[21] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2018. [Cited on page 4.]

[22] Izhar Wallach, Michael Dzamba, and Abraham Heifets. Atomnet: a deep convolutional neural network for bioactivity prediction in structure-based drug discovery. *arXiv preprint arXiv:1510.02855*, 2015. [Cited on page 1.]

[23] Yongji Wu, Defu Lian, Yiheng Xu, Le Wu, and Enhong Chen. Graph convolutional networks with markov random field reasoning for social spammer detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 1054–1061, 2020. [Cited on page 1.]

[24] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019. [Cited on pages 1, 10, 11, and 15.]

[25] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. [Cited on pages 5, 8, and 9.]

A Reproduction of ground truth dataset generation from [9]

The principal aim of this paper is to construct a ground truth explanation dataset, consisting of more than five million data points, for the HOMO-LUMO gap chemical property. To do that, authors discuss the use of atomwise and bondwise attributions, which are used to detect the most influential components of a molecule concerning a particular chemical property. The paper introduces chemical operations that enable the extraction of atomwise or bondwise attributions from existing chemical property datasets, without resorting to approximations or alchemical steps. This dataset fills a critical void in the domain of molecular graph explanation values, thereby aiding in the benchmark of various XAI methodologies.

The method used to generate atomwise and bondwise attribution from any dataset is defined as follows:

$$e(p, M, i) = p(M) - p(M'_i), \quad M'_i = \hat{o}(M, i) \tag{A.1}$$

where e is the explanation for a selected chemical property p, molecule M and atom or bond i. M'_i is the paired molecule defined by $\hat{o}(M, i)$, where $\hat{o}$ is a chemical operator on atom or bond i which modifies the initial molecule M. To ensure the explanation is only defined for exactly one atom or bond, the chemical operator only changes one atom or bond i, as detailed with two decision trees in [9].

By applying method A.1 on each atom and bond of a given dataset, one can therefore generate a dataset of ground truth explanations for a given property, e.g. HOMO-LUMO gap. The method has been run on the PCQM4Mv2 dataset and created two explanation datasets, respectively containing $3,619,970$ atomwise and $1,555,262$ bondwise explanation values. As seen in Fig. 11, the method does not always give an attribution for each bond or atom. This is due to the chemical transformation of a given molecule M which might not be listed in the reference dataset and can therefore not be compared to the original molecule. Needing a reference dataset for the paires molecules might become a constraint if we want to compare explanations for a large number of molecules.

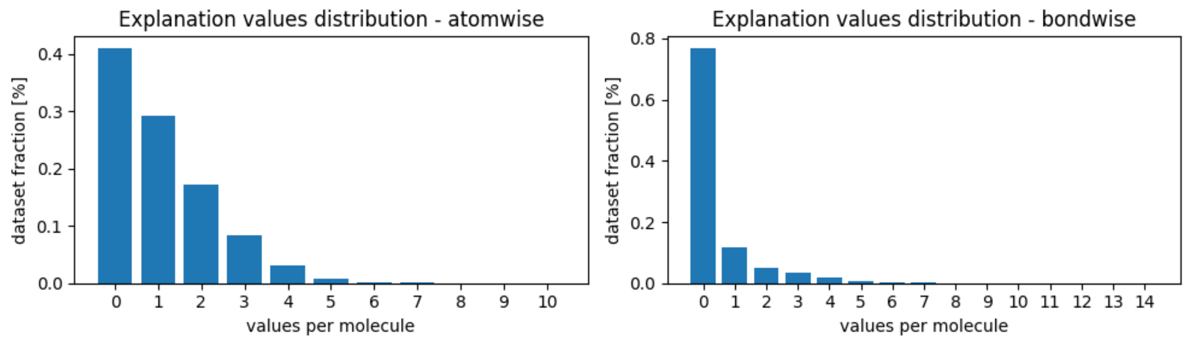


Figure 11: Distribution of the number of attributions per molecule in the PCQM4Mv2 dataset

Further discussions are also made on the relation between the HOMO-LUMO gap value and the type of atom for the atomwise attributions, as well as the dependence of the bondwise attribution on ring size. The assumption of smoothness for neighbouring attributions is also validated with the ground truth dataset by investigating the similarity of the explanation values for a pair of atoms (or bonds) in the same molecule depending on their closeness. Namely, it is shown that the autocovariance decreases strongly after a distance of two bonds for atomwise explanation and a distance of one bond for bondwise explanation values.

B Practical note on (un)directed graphs

In this project, and the libraries which are used, one defines an undirected graph as a graph having each edge becoming two directed edges with opposite directions. We will say that a given (un)directed graph contains $|E|$ edges but in reality, an undirected graph will be made with $\tilde{E}$ where $|\tilde{E}| = 2|E|$.

In Pytorch Geometric, which is the package that we used throughout this project, the authors defined *Data* object, which can be seen as dictionaries used to represent graphs. This "dictionary" contains attributes such as the *edge_index* which contains the $2 \times |\tilde{E}|$ adjacency list where each element is a node index, x is a $V \times d$ matrix containing the node d-dimensional features. Furthermore, for an undirected graph, *edge_attr* is a matrix of size $|\tilde{E}| \times d'$ which contains the edge features in dimensionality E_d.

Doubling each edge forced us in the creation of the regSHAPer algorithm to remove both edges at the same time, when constructing subgraphs. We chose to stay consistent with the meaning of "removing an edge" for a given graph, while the EdgeSHAPer ran its permutation-based sampling on the set $\tilde{E}$, therefore creating one-way connections between nodes. We do not know whether it has an important impact on the results, and this question should be discussed further. Lastly, when we had importance scores of size $2|E|$ for edge-centric explanations, we chose to convert it back to a size $|E|$ array by taking the mean of both edges between the same pair of nodes. Other methods can be used, such as taking the maximum or minimum of the two explanations.

C Methodology for experiments

The general experiment can be divided into three main parts, namely, training the GNN model, running the explainability methods and comparing the explanations.

C.1 Training the GNN model

From the source code of the model described in [10], we were able to train the model on the PCQM4Mv2 dataset containing PyG graphs. We first split the dataset into a training and validation set, respectively containing $3,378,606$ and $73,545$ graph molecules. We started training the GCN model on the training set, and used the validation set to assess the performance of the model using MAE. For each epoch, if the current MAE was better than the previous one, we would store the model into a *pytoch checkpoint* object.

Regarding hyperparameters for the GNN model, we set the drop ratio to zero, the number of layers to five, the dimensionality of hidden units in GNNs to 600, batch size to 256, and lastly the epochs to 100 but we had to cut the training process to 75 as previously stated.

We got a final validation MAE of 0.14, which is close to the one obtained by the authors with 100 epochs (0.1379).

C.2 Running the explainability methods

Once the model has been trained, we selected a list of molecules and tried to explain the model's prediction on those molecules. We ran the explainability methods for two days each (except GNNExplainer which was run for four days). This yielded 43000 explanations for GNNExplainer, 8100 for EdgeSHAPer and 11775 for regSHAPer. An important factor in our analysis was to deal with the size of the explanations. Namely, as discussed in B, each edge is encoded as two directed edges. Therefore, both GNNExplainer and EdgeSHAPer computed importance for each directed edge, while the regSHAPer was implemented to view the two directed edges as a single edge. To divide the length of the explanations from GNNExplainer and EdgeSHAPer, we decided to compute for each edge, the mean of its two directed edges. Other methods such as taking the maximum or minimum of the directed edges can also be used but will not be discussed here.

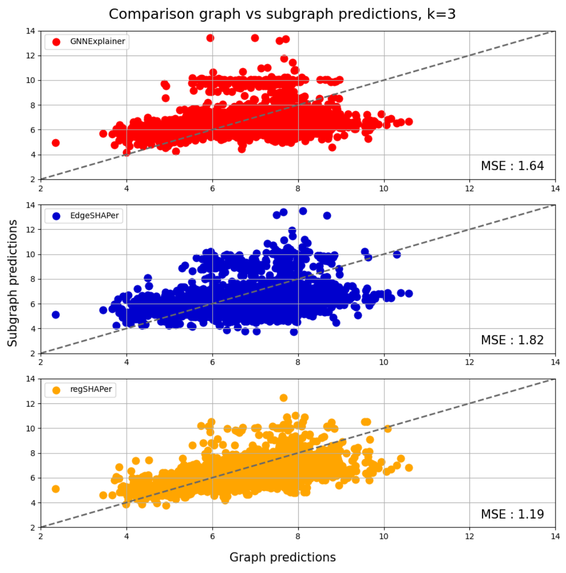
Once we had all those explanations with the same lengths for a given molecule, we started computing the various regression-based metrics, as well as, computing plots and visualizations of explanations, which have previously been described and analyzed in Section 6.

D Supplementary material

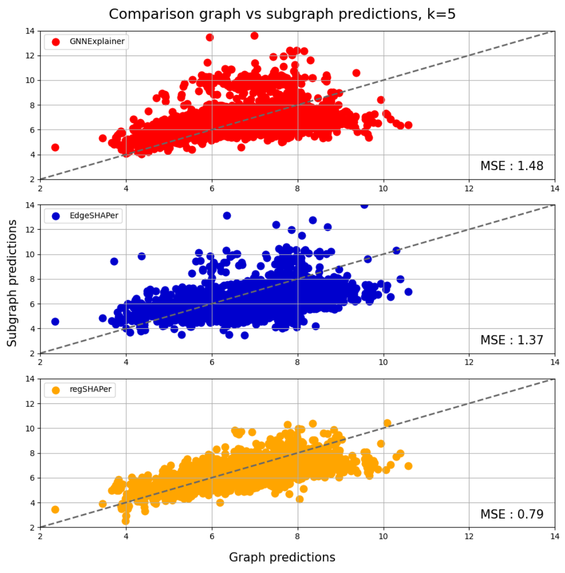
D.1 Prediction vs. *top-k* prediction, fidelity and faithfulness

Hereunder are presented a couple of graphs which show of the prediction vs. *top-k* prediction plots described in Section D.1 differ with respect to k. One observes in Figure 12 that the more we increase

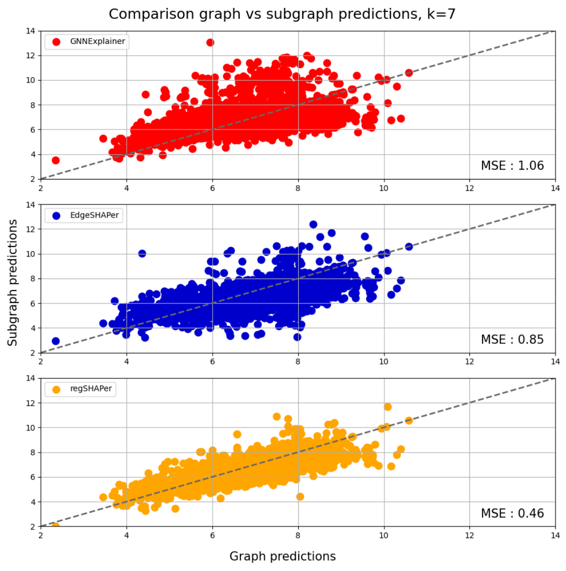
k, the lower the MSE, as we let the *top-k* graph take more edges for its subgraph creation.



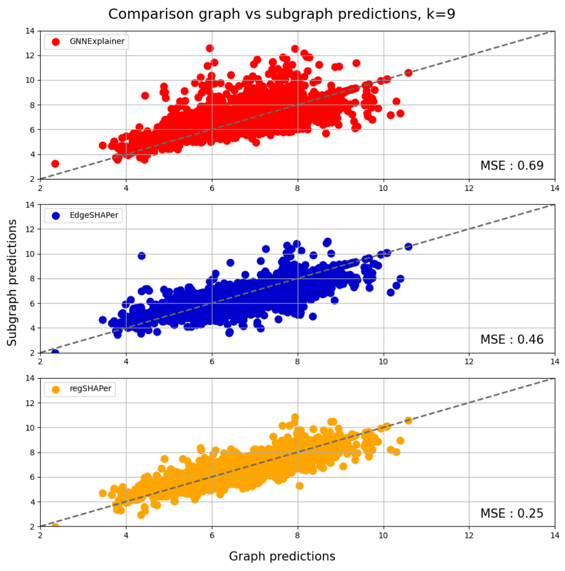
(a) $k = 3$



(b) $k = 5$



(c) $k = 7$

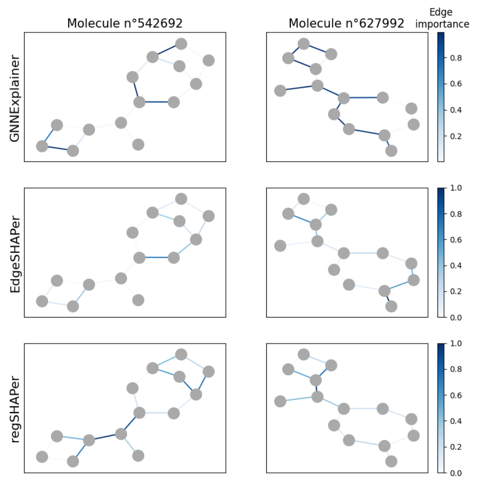


(d) $k = 9$

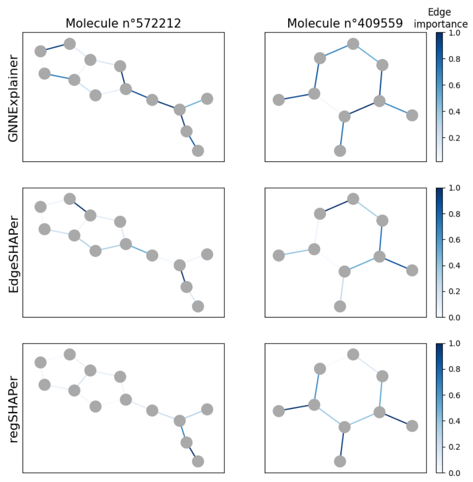
Figure 12: Comparison of the prediction vs *top-k* prediction graphs when varying k

D.2 Randomly picked explanation visualizations

On Figure 13 are presented a couple of graphs which show additional explanation visualizations plots for randomly picked molecules from the PCQM4Mv2 dataset.



(a) Molecule n°542, 692 (left), and n°627, 992 (right)



(b) Molecule n°572, 212 (left), and n°409, 559 (right)

Figure 13: Additional visualizations of explanations with respect to explainability methods

As seen in Figure 13, we still observe the lack of sparsity for the GNNExplainer, especially for molecules 627, 992 and 572, 212. The sparsity of the regSHAPer and edgeSHAPer are equivalent, and high. The second insight is the difference of importance patterns for each explainability, as nearly each edge has different importance depending on with explainer is used. Further steps would be to compare those explanations to ground truth, and check whether those different masks are of good quality.