Scalable GNN Explanations with Distributed Shapley Values

Selahattin Akkas
Indiana University Bloomington
Indiana, USA
sakkas@iu.edu

Aditya Devarakonda
Wake Forest University
North Carolina, USA
devaraa@wfu.edu

Ariful Azad
Texas A&M University
Texas, USA
ariful@tamu.edu

ABSTRACT

With the growing adoption of graph neural networks (GNNs), explaining their predictions has become increasingly important. However, attributing predictions to specific edges or features remains computationally expensive. For example, classifying a node with 100 neighbors using a 3-layer GNN may involve identifying important edges from millions of candidate subgraphs. To address this challenge, we develop DistShap, a parallel algorithm that distributes Shapley value-based explanations across multiple GPUs. DistShap samples subgraphs in a distributed setting, executes GNN inference in parallel across GPUs, and solves a distributed least squares problem to compute edge importance scores. DistShap outperforms most existing GNN explanation methods in accuracy and is the first to scale to GNN models with millions of edges by using 128 GPUs.

PVLDB Reference Format:
Selahattin Akkas, Aditya Devarakonda, and Ariful Azad. Distributed GNN Explanations. PVLDB, 19(8): 1731 - 1739, 2026.
doi:10.14778/3811243.3811247

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/HipGraph/DistShap.

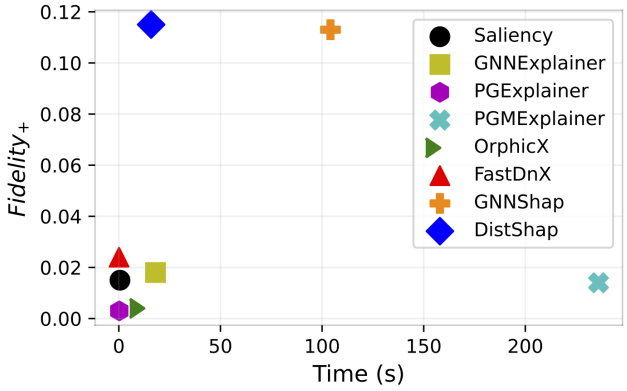

Figure 1: $Fidelity_+$ scores for identifying the top 20 most influential edges and corresponding computation times on the Coauthor-CS graph for seven explanation methods. A two-layer GCN was pre-trained before the explanation task. DistShap was run on 8 GPUs, while other methods used a GPU. Higher $Fidelity_+$ scores and lower runtimes are desired.

1 INTRODUCTION

Over the past decade, Graph Neural Networks (GNNs) have gained popularity for various learning tasks on graph-structured data. Despite methodological differences, most popular GNNs [11, 15, 32] share a common computational pattern: at each layer, a node aggregates information from its neighbors and applies a neural transformation to the aggregated representation. Hence, a node's prediction depends on a complex aggregation of its multi-hop neighbors, making it challenging to interpret the contributions of individual nodes, edges, and features to the prediction. GNN explanation methods aim to address this challenge.

GNN explanation. We consider a black-box explanation setting in which the objective is to understand how a trained GNN arrives at its predictions during inference. Consider a two-layer GCN as an example: the prediction for a node v is influenced by all nodes within its two-hop neighborhood (called the computational graph). However, these neighbors do not contribute equally; some have a substantially greater impact on the prediction than others. A GNN explainer aims to identify which edges are most influential for the prediction of v. Formally, an explanation method returns a subgraph G_s induced by a subset of v's two-hop neighborhood, where the edges in G_s are deemed most important for the prediction of v.

Effectiveness of explanations. Let G_s be the subgraph containing edges important for the prediction of v. This explanation is considered effective if removing G_s from the original graph significantly reduces the model's prediction score. This measure, known as $Fidelity_+$ [36], captures the idea that edges are important if their removal negatively impacts the model's ability to make accurate predictions. A high $Fidelity_+$ score indicates an effective explanation, while a low score suggests it is uninformative.

The need for scalable GNN explanations. We observed that the effectiveness of existing explanation methods [34–36] decreases for nodes with large computational graphs. For example, a node with 1,000 edges would require evaluating 2^{1000} subgraphs to compute the importance of each edge, an infeasible task. Most methods trade accuracy for speed by relying on sampling or surrogate models. However, as shown in Fig. 1, approximation enables most existing methods to run faster but results in low $Fidelity_+$ scores, whereas GNNShap [2] attains high fidelity at the cost of a much higher runtime. Given GNNShap's high fidelity but slow runtime, a scalable Shapley-based explanation method is needed to preserve fidelity while significantly accelerating computation.

Contributions. The main contribution of this paper is DistShap, the first distributed explanation method for GNNs that enables significantly faster computations. Similar to GNNShap [2]

Proceedings of the VLDB Endowment, Vol. 19, No. 8 ISSN 2150-8097.
doi:10.14778/3811243.3811247

and GraphSVX [7], DistShap employs approximate Shapley values but overcomes their high computational and memory costs by distributing the workload across multiple GPUs. DistShap consistently matches the accuracy of the state-of-the-art GNNShap while enabling explanations of large computational graphs using tens of GPUs. Our key contribution lies in achieving scalability without compromising accuracy.

2 BACKGROUND & RELATED WORK

We consider a graph $G(V, E)$, where V represents the set of nodes and E represents the set of edges. Let $A \in \mathbb{R}^{|V|\times|V|}$ be the sparse adjacency matrix of the graph, where $A_{ij} = 1$ if $v_i, v_j \in E$, and $A_{ij} = 0$ otherwise. Additionally, let $X \in \mathbb{R}^{|V|\times d}$ be the matrix of d-dimensional node features. Each node is assigned to one of $\mathbb{C}$ classes for node classification tasks. Given a trained GNN model f, the predicted class for a node v is $\hat{y} = f(G, X, v)$.

2.1 Graph Neural Networks

In a message-passing GNN, the lth layer performs three key computations [40]. First, messages are propagated between node pairs (v_i, v_j) based on their previous layer representations h_i^{l-1} and h_j^{l-1}: $q_{ij}^l = \text{MESSAGE}(h_i^{l-1}, h_j^{l-1})$. Next, the messages from neighbors $\mathcal{N}_{v_i}$ are aggregated: $q_i^l = \text{AGGREGATE}(\{q_{ij}^l | v_j \in \mathcal{N}_{v_i}\})$. Finally, the GNN updates the node representation h_i^l by applying a non-linear transformation to the aggregated message and the node's previous representation: $h_i^l = \text{UPDATE}(q_i^l, h_i^{l-1})$.

Computational graph: Once a GNN model f is trained, we can predict the class of a test node v as $\hat{y} = f(G, X, v)$. This prediction does not require access to the entire graph. Specifically, a GNN with L layers only depends on v's 1-hop through L-hop neighbors, the edges among them, and any associated node and edge features. This L-hop subgraph is referred to as the computational graph $G_c(v)$, which contains all the necessary information for predicting v.

2.2 GNN Explanation

To explain the prediction of node v, a GNN explanation method takes a trained GNN model f and the computational graph $G_c(v)$ of node v as inputs. It then outputs a small subgraph $G_s(v)$ of $G_c(v)$ as the output. $G_s(v)$ contains only a few important nodes and edges, thereby providing insight into the reasoning behind the prediction.

2.3 Shapley Values

Shapley values [28] come from cooperative game theory and are used to fairly assign credit to each player (e.g., a feature or graph edge) for their contribution to the outcome of a game (e.g., a model prediction). Let the game be defined by a set of n players $N = \{1, 2, ..., n\}$ and let $f : 2^N \to \mathbb{R}$ be a value function that assigns a real number (e.g., model prediction) to each coalition $S \subseteq N$. The Shapley value ϕ_i for player $i \in N$ is defined as:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} \left[f(S \cup i) - f(S) \right], \tag{1}$$

where $f(S \cup i) - f(S)$ is the marginal contribution of player i to the coalition S. Therefore, Eq. 1 computes the weighted average of i's marginal contributions across all possible coalitions that exclude i. For GNN explanations, edges in the computational graph are considered players, and each subgraph represents a coalition of these players. Therefore, a computational graph with n edges can generate 2^{n-1} possible subgraphs or coalitions.

Computing exact Shapley values is infeasible for large numbers of players. Kernel SHAP [17] addresses this by approximating Shapley values using an additive linear surrogate model, where the sum of the Shapley values equals the model prediction. The surrogate model g is defined as:

$$g(x) = \phi_0 + \sum_{i=1}^{n} \phi_i m_i, \tag{2}$$

where $m \in \{0, 1\}^{1\times n}$ is a binary coalition mask that makes a coalition S, and ϕ is the surrogate model's parameters. The model parameters are the approximation of the Shapley values. $\phi_0 = f(\emptyset)$ is the case when there are no players. In addition, when a player is missing ($m_i = 0$), the corresponding input of the model should be replaced with background data (e.g., expected value for the player). This model is solved by the following weighted linear regression:

$$\min_{\phi} \sum_{S \subseteq N} w(S) \left[f(S) - (g(S)) \right]^2, \tag{3}$$

where the kernel weight is defined as follows

$$w(S) = \frac{n - 1}{\binom{n}{|S|} |S|(n - |S|)}. \tag{4}$$

In this setting, the kernel weight assigns individual coalition weights based on coalition size, giving more weight to smaller and larger coalitions due to the ease of observing individual effects.

Equation 3 samples k coalitions from the 2^n possible subsets of n players. Let $M \in \{0, 1\}^{k\times n}$ be a binary mask matrix, where $M[i, j] = 1$ if player j is included in the ith coalition . Let W be a diagonal matrix of sample weights, and $\hat{y} \in \mathbb{R}^k$ the corresponding model predictions. Then, Equation (3) can then be written as

$$\min_{\phi} \sum_{i=1}^{k} w_i \left(\hat{y}_i - m_i \phi \right)^2,$$

where w_i is the i-th diagonal element of W and m_i is row i of M. Writing this in matrix-form, yields

$$\min_{\phi} \left\| \sqrt{W} \left(\hat{y} - M\phi \right) \right\|_2^2$$

which is the weighted least squares problem [1] whose solution is given by:

$$\phi = (M^T W M)^{-1} M^T W \hat{y} \tag{5}$$

The remaining challenge is sampling coalitions from 2^n subsets. Kernel SHAP [17] distributes samples by weighting coalition sizes. Let $\rho_{|S|}$ be the total weight of all coalitions of size $|S|$; the number of samples $k_{|S|}$ drawn at size $|S|$ is then computed as follows:

$$\rho_{|S|} = \frac{n - 1}{|S|(n - |S|)}, \qquad k_{|S|} = k * \frac{\rho_{|S|}}{\sum_{i=1}^{n-1} \rho_i} \tag{6}$$

DistShap adopts this non-uniform sampling strategy.

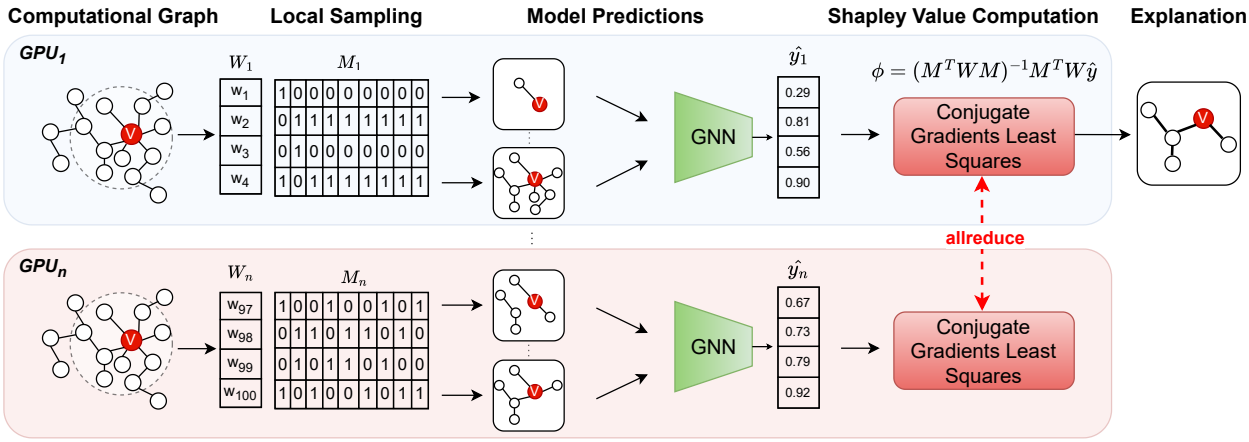

Figure 2: DistShap computes Shapley values for all edges in the computational graph of a target node v, using a two-layer GCN model as an example. First, the computational graph of v is replicated across multiple GPUs. Each GPU independently generates a subset of samples using its local mask matrix and performs model predictions for v on the corresponding sampled subgraphs. Once all predictions are completed, the GPUs collaboratively compute approximate Shapley values for every edge in the computational graph. Finally, DistShap identifies and returns the top-k most important edges based on Shapley values.

2.4 Related Work

Most GNN explanation methods [36] aim to explain how a GNN makes a specific prediction, such as a node classification. These methods fall into three main categories. *Gradient-based methods* use gradients, activations, or attention weights to estimate importance. Examples include Saliency[4, 26], Guided Backpropagation [4], CAM, GradCAM [26], and Integrated Gradients [30].

Perturbation-based methods assess importance by altering nodes, edges, or subgraphs and measuring the impact on predictions. Examples include GNNExplainer [34], PGExplainer [18], GraphMask [27], SubgraphX [37], EdgeSHAPer [20], GraphSHAP [25], GStarX [38], FlowX [9], and Zorro [8].

Surrogate-based methods train a simpler, inherently interpretable surrogate model (e.g., a linear model) to approximate the GNN's behavior around a specific prediction. Examples include GraphLime [14], RelEx [39], GraphSVX [7], PGM-Explainer [33], FastDnX [24], GNNShap [2]. Their effectiveness depends on how well the surrogate mimics the GNN.

The aforementioned methods focus on factual reasoning, identifying subgraphs sufficient to reproduce the original prediction. Several recent studies also explore counterfactual reasoning, which seeks subgraphs whose removal alters the prediction [31].

Game-Theoretic methods. GRAPHSHAP [25] and EdgeSHAPer [20] focus on graph classification by assigning importance scores to edges or graph motifs. GraphSVX [7] explains node classification by using an approximation strategy that heavily samples small and large coalitions, potentially leading to estimations of suboptimal explanations. Similar to GraphSVX, GNNShap [2, 3] estimates Shapley values for edges by sampling coalitions. SubgraphX [37] aims to identify the most explanatory subgraph using Shapley values, employing a Monte Carlo Tree Search (MCTS) to explore possible subgraphs. GStarX [38] also uses MCTS but utilizes Hamiache-Navarro [10] values.

Distributed Shapley Values. Several works have explored distributed Shapley value computation. For example, [19] uses Apache Spark but is limited to tens of players, while GPUTreeShap [21] supports multi-GPU execution only for tree-based models. In contrast, our work targets GNN explanations with thousands of players.

3 DISTRIBUTED SHAPLEY VALUES

3.1 Overview of DistShap

A Shapley-based GNN explanation typically involves five steps: **(1) Define the game**. For GNN explanations, players are edges in the computational graph $G_c(v)$, coalitions are subgraphs of $G_c(v)$, and the value function is the output of the trained GNN model f. **(2) Sample coalitions.** For GNN explanation, each sampled coalition is a subgraph of $G_c(v)$. **(3) Model prediction.** For each sampled subgraph or coalition, we compute the output of the GNN model. **(4) Estimate Shapley values.** We then estimate Shapley values by solving the weighted least squares problem. **(5) Rank and select important edges.** The final Shapley values indicate the importance of each edge of the computational graph. We select the top-k important edges to form the explanation subgraph.

DistShap parallelizes the sampling, model prediction, and Shapley value computation across multiple GPUs. Figure 2 shows an overview of DistShap for the task of explaining the prediction of node v. We describe each step in the following sections.

3.2 Distributed Subgraph Sampler

3.2.1 Replicating the computational graph. We extract the computational graph $G_c(v)$ for the target node v and replicate it on each GPU. While the full graph may exceed GPU memory, individual computational graphs are small and easily fit (Table 4). Replication allows independent sampling and prediction on each GPU without communication, enabling near-linear scalability of DistShap.

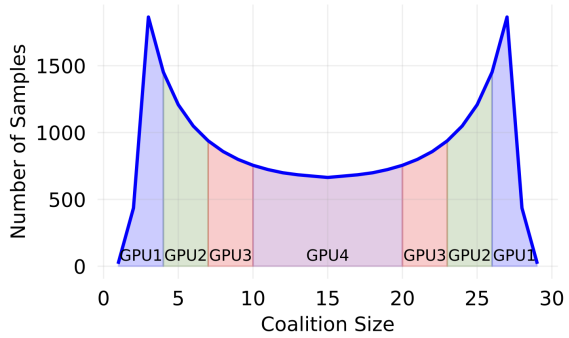

Figure 3: DistShap distribution of 25,000 samples among 4 GPUs. Here, the computational graph has 30 edges, and hence the coalition size ranges from 1 to 29.

3.2.2 Distributed Subgraph Sampling. In the sampling step, a subgraph (i.e., a coalition) is randomly selected from the computational graph. In this step, two decisions must be made: (1) how to generate k representative samples from the exponentially large space of 2^n subgraphs, and (2) how to efficiently distribute k samples across p GPUs in a computationally efficient and load-balanced manner.

Non-uniform Sampling. Prior work [2, 7] has shown that uniform sampling from the 2^n possible subgraphs does not provide good explanations. Hence, we adopt a non-uniform sampling strategy guided by Eq. 6 and Fig. 3. This distribution favors sampling coalitions with very few or very many edges, as these tend to provide more informative signals for explaining the prediction. However, mid-sized coalitions are not ignored, ensuring a balanced coverage across the coalition space, as shown in Fig. 3. Moreover, the sampling strategy is symmetric: for every sampled subgraph with a edges, its complementary subgraph with $n-a$ edges is also included. This pairing of complementary samples helps the algorithm converge more quickly to the true Shapley values.

Partitioning samples across GPUs. To generate k samples across p GPUs, each GPU should handle k/p samples for balanced computations. However, a naive sequential assignment of samples to GPUs can cause load imbalance, as some samples (e.g., those with fewer edges) require less computation. To address this, we exploit sampling symmetry: each GPU generates both samples in a complementary pair, as illustrated in Fig. 3. Since each pair of complementary subgraphs contains a total of n edges, this strategy ensures that each GPU processes k/p samples and exactly $\frac{nk}{2p}$ edges, achieving perfect load balance.

Sampling via a mask matrix. Instead of sampling subgraphs directly, we generate a $k \times n$ binary mask matrix M, where each row defines the edges included in a sample ($M[i, j] = 1$ means edge j is present in sample i). Due to paired sampling, M is 50% sparse. Each GPU generates k/p rows of M, which reduces to sampling from a Bernoulli distribution, an embarrassingly parallel task ideal for GPUs. In practice, we generate only $k/2$ samples and derive their complements deterministically. Subgraphs are created by masking the edge list with M, enabling highly parallel and efficient sampling, as shown in Fig. 2 and Fig. 4.

3.2.3 Time and Memory Complexity. The memory required to store the submatrix of M and corresponding subgraphs is $O(n \cdot k/p)$ per GPU. As t threads in a GPU can generate samples independently, the time complexity is $O(\frac{n \cdot k}{t \cdot p})$.

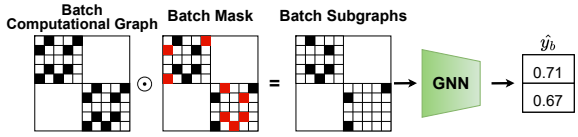

Figure 4: Batch masking and model prediction by stacking the computational graph diagonally. Red cells show edges that are not in the coalition.

3.3 Distributed Model Prediction

We perform GNN predictions for each sampled subgraph. For example, with 600K samples across 10 GPUs, each GPU handles 60K predictions. Running these sequentially underutilizes the GPU, so we batch subgraphs for parallel inference. As shown in Fig. 4, we construct a block-diagonal adjacency matrix and a matching block-diagonal mask matrix. Their element-wise product yields the sampled subgraphs, enabling the GNN to run inference on the entire batch at once. Batching improves parallel efficiency but comes at the cost of increased memory usage.

Time and memory complexity. Suppose we are using a two-layer GCN with a hidden dimension of d. Let b be the number of batches, and $|V_c|$ be the number of nodes in the computational graph. Then, the time for batched model prediction on each GPU is

$$O\left(b \cdot (n \cdot d + |V_c| \cdot d^2)\right). \tag{7}$$

The memory requirement for a batch is

$$O\left(b \cdot (n + |V_c| + |V_c| \cdot d)\right). \tag{8}$$

Increasing the batch size leads to a linear increase in GPU memory usage. However, it also improves GPU utilization, thereby reducing overall computation time (up to the point where resource limits are reached). While memory usage can be estimated using Eq. 8, runtime behavior across batch sizes is less predictable. We conducted an empirical study (Fig. 8) and found that a batch size of 50 provides the best balance between runtime and memory usage.

3.4 Distributed Shapley Computations

Shapley values are the solution to the following weighted least-squares problem: $\phi = (M^T W M)^{-1} M^T W \hat{y}$, where M is the mask matrix, W is a diagonal weights matrix, and $\hat{y}$ is the vector of model predictions. Since $k \gg n$, a parallel direct method to obtain ϕ is to store M in a 1D-row layout and perform the following computations in parallel: compute $G = M^T W M$, $r = M^T W \hat{y}$ and solve the linear system $\phi = G^{-1} r$ sequentially. Communication is required to sum-reduce G across all GPUs, which requires communicating n^2 entries. However, for large enough n, solving the linear system sequentially and communicating all of G is likely to become a bottleneck. Other direct methods based on LU, QR, or their tall-skinny variants (e.g., TSQR [6]) require complex 2D block cyclic data layouts (e.g., LU and QR) and also require parallel triangular solve routines (LU, QR, and TSQR), which are challenging to parallelize and scale.

Algorithm 1 Distributed Conjugate Gradients Least-Squares

Input: $M_p \in \mathbb{R}^{k/p \times n}, W_p \in \mathbb{R}^{k/p \times k/p}, \hat{y_p} \in \mathbb{R}^{k/p}, x_0 \in \mathbb{R}^n$ ▷ Subscript p denotes GPU-local inputs
$M_p = \sqrt{W_p} M_p$
$\hat{y_p} = \sqrt{W_p} \hat{y_p}$
$r_0 = \hat{y_p}$
$s_0 = M_p^\intercal \hat{y}$
$s_0 = \text{ALL\_REDUCE}(s_0)$ ▷ n-dimensional vector reduction
$u_0 = s_0$
for $i = 1, 2 \ldots$ until convergence **do**
  $v_k = M_p u_{i-1}$
  $\delta = \|v_i\|_2^2$
  $\delta = \text{ALL\_REDUCE}(\delta)$ ▷ Scalar reduction
  $\theta = \frac{\|s_{i-1}\|_2^2}{\delta}$
  $x_i = x_{i-1} + \theta u_{i-1}$
  $r_i = r_{i-1} - \theta v_{i-1}$
  $s_i = M_p^\intercal r_i$
  $s_i = \text{ALL\_REDUCE}(s_i)$ ▷ n-dimensional vector reduction
  $p_i = s_i + \frac{\|s_{i-1}\|_2^2}{\|s_i\|_2^2} u_{i-1}$
end for
return x_i

Instead, we utilize a distributed-memory implementation of the Conjugate Gradients Least Squares (CGLS) method [12, 22, 23] to solve the weighted least-squares problem iteratively. CGLS is a Krylov method that adapts CG to the least squares problem by implicitly solving the normal equations, and is more numerically stable than applying a parallel CG algorithm to solve $G^{-1}r$. Each iteration of CGLS requires a sequence of matrix-vector, vector, and scalar operations that are simpler to parallelize. While Krylov methods are often applied to solve sparse linear systems, we apply CGLS to solve a dense least-squares problem. Since M has a sparsity of 50%, we store it in a dense format, which enables us to use the more efficient dense matrix-vector multiplication kernel on the GPU instead of an SpMV. We parallelize CGLS by storing M in a 1D row layout such that each GPU stores k/p rows of M locally. We partition $\hat{y}$ similarly but replicate all n-dimensional vectors used in CGLS. Under a 1D row layout, norm computations of k-dimensional vectors and matrix-vector products with $M^\intercal$ require reductions across GPUs to ensure sequential consistency.

We show the distributed CGLS algorithm in Algorithm 1, where each iteration performs orthogonalization of the solution (x_i) and residual (r_i) vectors associated with the left and right subspaces of M. Scalar quantities (e.g. θ) represent step-sizes use to update the conjugate directions s_i and p_i. Without round-off error, CGLS requires n iterations to find the solution ϕ. However, in practice, the number of iterations depends on the condition number of the input matrix, M. Communication is required every iteration to compute δ and s_i. All remaining operations are local to each GPU. The communication complexity associated with Algorithm 1 is

$$T_{comm} = \alpha \cdot O(i \cdot \log p) + \beta \cdot O(i \cdot n),$$

where i is the number of iterations to find the solution, α and β are hardware latency and inverse bandwidth parameters, respectively.

Table 1: Dataset Summary

Dataset	Nodes	Edges	Features	Classes
Coauthor-CS	18,333	163,788	6,805	15
Coauthor-Physics	34,493	495,924	8,415	5
DBLP	17,716	105,734	500	4
ogbn-arxiv	169,343	2,315,598	128	40
ogbn-products	2,449,029	123,718,280	100	47
Reddit	232,965	114,615,892	602	41

4 EXPERIMENT SETTINGS

Datasets. We use six datasets for our experiments: Coauthor-CS, Coauthor-Physics [29], DBLP [5], Reddit [11], ogbn-arxiv and ogbn-products [13]. These datasets include co-author, citation, and co-purchase networks, with nodes representing authors, papers, products, or posts. Node features range from keywords and bag-of-words to skip-gram embeddings. Table 1 summarizes these graphs.

GNN Models. All explanation methods studied here are model-agnostic and can be applied to any GNN. We experimented with a two-layer GCN trained for node classification, with 128 hidden dimensions for Reddit, ogbn-products, and ogbn-arxix, and 64 for Coauthor and DBLP. We also experimented with a 2-layer graph attention network (GAT) with 8 attention heads. The models are trained with cross-entropy loss.

Explanation Setting. We select 50 test nodes for explanation. For Reddit, ogbn-arxiv, and ogbn-products, nodes have 50K–100K edges in their computational graphs. For Coauthor and DBLP, we choose nodes with at least 1K edges in their computational graphs.

Test Environment. Experiments are run on the Perlmutter supercomputer at NERSC, which features nodes with 2 AMD EPYC 7713 CPUs, 512 GB of RAM, and 4 NVIDIA A100 (80 GB) GPUs. We use Python 3.12.3, CUDA 12.4, PyTorch 2.4.1, and PyG 2.6.0.

Evaluation Metrics. $Fidelity_+$ and $Fidelity_-$ are widely used metrics [36] to evaluate explanation quality. Given a node v, let $G_c \subseteq G$ be its computational graph and $G_s \subseteq G_c$ the subgraph identified as important. Then, $Fidelity_+$ for node v is computed as:

$$Fidelity_+(v) = |f(G_c) - f(G_c \setminus G_s)| \tag{9}$$

Here, $f(G_c)$ is the model's prediction score for node v on the full graph, and $f(G_c \setminus G_s)$ is the score obtained after removing the important edges in G_s. Eq. 9 shows the calculation of $Fidelity_-$.

$$Fidelity_-(v) = |f(G_c) - f(G_s)| \tag{10}$$

$Fidelity_+$ measures the impact of removing important edges, while $Fidelity_-$ measures the effect of removing unimportant ones. Good explanations have high $Fidelity_+$ and low $Fidelity_-$ scores.

GNN Explanation Baselines. We used 8 baseline methods to compare DistShap. Saliency [4, 26] uses input gradients to estimate node importance. GNNExplainer [34] identifies key edges and features via mutual information. PGExplainer [18] learns explanations using a neural network trained with mutual information. PGMExplainer [33] applies probabilistic graphical models to assess node importance. OrphicX [16] generates causal explanations with a generative surrogate model. FastDnX [24] explains GNNs using linear surrogate model decomposition. GraphSVX [7] is a Shapley value method that considers nodes as players. GNNShap [2] utilizes GPU for fast sampling and model predictions.

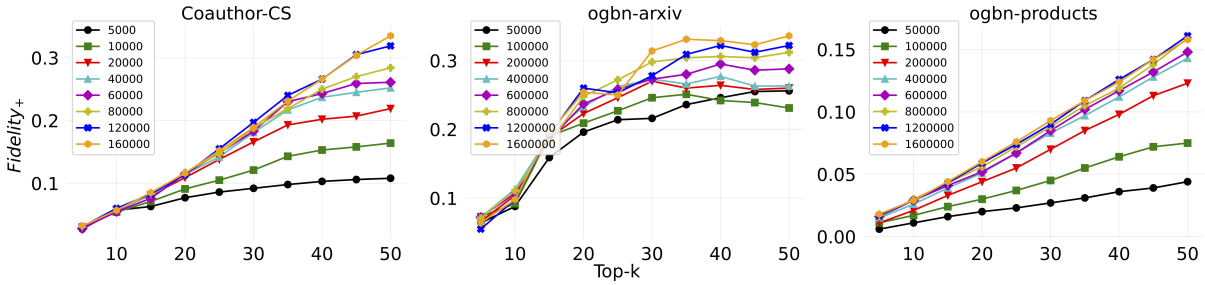

Figure 5: Average $Fidelity_+$ scores (higher is better) for the top-k most important edges identified by DistShap across various sample sizes. Increasing the number of samples generally improves the results, up to a point of saturation. .

Table 2: At the top, we show $Fidelity_+$ scores for top-k most important edges (higher is better). At the bottom of the table, we show $Fidelity_-$ scores across different sparsity levels (lower is better), where a sparsity of 0.6 indicates 60% of the least important edges are removed. In all cases, a 2-layer GCN is used, and the average Fidelity score is computed over 50 test nodes. DistShap and GNNShap used 60K samples for small graphs and 600K samples for large graphs. Bold values indicate the best results. DistShap achieves the best or near-best performance across all settings. OOM means out of memory.

		Coauthor-CS		Coauthor-Phy		DBLP		ogbn-arxiv		ogbn-products		Reddit	
	topk	10	30	10	30	10	30	10	30	10	30	10	30
$Fidelity_+$	Saliency	0.006	0.03	0.001	0.003	0.005	0.018	0.008	0.013	0.001	0.002	0.011	0.024
	GNNExplainer	0.007	0.032	0.001	0.067	0.036	0.165	0.033	0.095	0.015	0.042	0.049	0.085
	PGExplainer	0.001	0.003	0.001	0.001	0.004	0.007	0.002	0.004	0.001	0.001	0.001	0.001
	PGMExplainer	0.009	0.016	0.004	0.007	0.013	0.027	0.021	0.023	0.002	0.002	0.011	0.017
	OrphicX	0.002	0.004	OOM	OOM	0.002	0.004	OOM	OOM	OOM	OOM	OOM	OOM
	FastDnX	0.011	0.043	0.001	0.011	0.015	0.052	0.044	0.084	OOM	OOM	OOM	OOM
	GraphSVX	0.002	0.004	0.000	0.001	0.002	0.005	OOM	OOM	OOM	OOM	OOM	OOM
	GNNShap	**0.060**	**0.191**	0.017	**0.075**	**0.062**	**0.203**	OOM	OOM	OOM	OOM	OOM	OOM
	DistShap	**0.060**	**0.191**	**0.019**	0.073	0.061	0.199	**0.105**	**0.291**	**0.030**	**0.086**	**0.099**	**0.202**
	sparsity	0.3	0.6	0.3	0.6	0.3	0.6	0.3	0.6	0.3	0.6	0.3	0.6
$Fidelity_-$	Saliency	0.026	0.054	0.002	0.008	0.01	0.025	0.013	**0.015**	0.011	0.043	0.006	0.018
	GNNExplainer	0.026	0.046	0.004	0.014	0.032	0.06	0.055	0.055	0.061	0.085	0.126	0.147
	PGExplainer	0.017	0.037	0.040	0.140	0.108	0.263	0.047	0.053	0.062	0.157	0.056	0.127
	PGMExplainer	0.014	0.025	0.005	0.011	0.012	0.029	0.032	0.051	**0.005**	**0.010**	0.005	0.022
	OrphicX	0.050	0.295	OOM	OOM	0.140	0.271	OOM	OOM	OOM	OOM	OOM	OOM
	FastDnX	0.022	0.063	0.006	0.023	0.019	0.048	0.01	0.022	OOM	OOM	OOM	OOM
	GraphSVX	0.065	0.141	0.059	0.079	0.104	0.202	OOM	OOM	OOM	OOM	OOM	OOM
	GNNShap	0.006	**0.012**	**0.001**	**0.002**	**0.007**	**0.016**	OOM	OOM	OOM	OOM	OOM	OOM
	DistShap	**0.005**	0.013	**0.001**	**0.002**	**0.007**	0.020	**0.006**	0.017	**0.005**	0.014	**0.003**	**0.006**

5 RESULTS

5.1 Impact of Sampling on Explanation Fidelity

We analyze the effect of sample size on DistShap's ability to identify important edges. Figure 5 reports average $Fidelity_+$ scores on Coauthor-CS, ogbn-arxiv, and ogbn-products. Fidelity improves with more samples but plateaus around 60K for Coauthor-CS and 600K for the larger graphs. We therefore use these sample sizes to balance accuracy and efficiency.

5.2 Fidelity Comparisons with Other Baselines

Table 2 reports average $Fidelity_+$ and $Fidelity_-$ scores for 50 test nodes using a 2-layer GCN. When finding top-10 and top-30 most important edges, DistShap consistently achieves the highest or second highest $Fidelity_+$ across all datasets. DistShap also gives the best or near-best results for $Fidelity_-$. PGMExplainer performs better on ogbn-products but lacks robustness on other datasets. Overall, DistShap performs comparably to GNNShap on small datasets, which is expected since DistShap is conceptually based on GNNShap. However, as shown in Table 2, GNNShap and GraphSVX – two Shapley-value-based methods – cannot handle large graphs due to their high computational and memory demands.

5.3 Explanation Runtime Comparison

Table 3 reports total explanation time for 50 test nodes. DistShap uses 128 GPUs for large datasets and 8 for smaller ones, while others

Table 3: Total GCN explanation times for 50 nodes. Training times of surrogate models are provided in parentheses. DistShap uses 8 GPUs for Coauthor and DBLP and 128 GPUs for other datasets. (Err: CUDA insufficient resources; OOM: Out Of Memory)

	Coauthor-CS	Coauthor-Phy	DBLP	ogbn-arxiv	ogbn-products	Reddit
Saliency	0.56	0.76	0.47	0.98	0.87	1.37
GNNExplainer	18.05	18.59	15.97	61.30	79.01	1,092.40
PGExplainer	0.22 (392.49)	0.24 (140.72)	0.23 (35.55)	0.85 (236.38)	1.54 (700.21)	14.83 (3924.19)
PGMExplainer	236.02	409.17	226.17	7,481.89	3,457.25	6,392.86
OrphicX	12.35 (1946.44)	OOM	16.15 (528.27	OOM	OOM	OOM
FastDnX	0.11 (66.36)	0.16 (285.73)	0.08 (10.20)	0.61 (21.55)	Err	Err
GraphSVX	11,828.21	22,059.34	12,704.41	OOM	OOM	OOM
GNNShap	104.73	179.33	33.24	OOM	OOM	OOM
DistShap	15.91	28.56	5.26	129.31	106.48	171.21

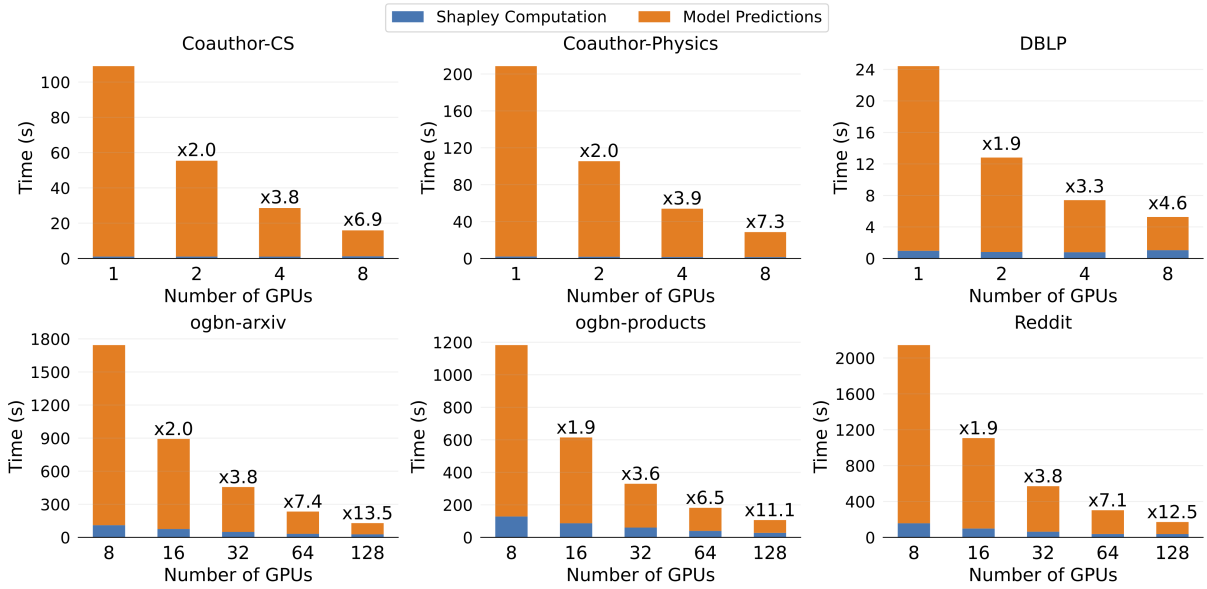

Figure 6: Scalability of DistShap across different numbers of GPUs. The y-axis denotes the time needed to explain predictions for 50 test nodes. The speedup values above each bar indicate performance improvements relative to the first bar for each respective dataset. We use 600,000 samples for large graphs and 60,000 for small ones.

use a single GPU. Although not a direct comparison, it highlights the efficiency of distributed Shapley value computation.

Overall, Saliency is the fastest method but exhibits poor fidelity, as shown in Table 2. PGExplainer and GNNExplainer are also relatively fast but produce subpar fidelity scores. PGMExplainer performs slowly and struggles particularly with $Fidelity_+$. As discussed earlier, GNNShap fails on large graphs due to memory constraints, while GraphSVX, another Shapley-based method, is several orders of magnitude slower than all other approaches considered. In contrast, DistShap delivers accurate explanations within practical runtimes by leveraging distributed computing.

5.4 Explaining the Prediction of GAT

Similar to the GCN experiments, we evaluate explanation quality using a 2-layer Graph Attention Network (GAT) with 8 attention heads. As in the GCN setting, GNNShap and DistShap achieve the best fidelity scores. However, GNNShap fails to scale to large datasets due to memory limitations. Due to space constraints, we present these results on the companion website.

5.5 Scalability of DistShap

Fig. 6 shows DistShap's runtime breakdown (sampling + prediction vs. Shapley value computation) across GPUs. For smaller graphs (top row), runtime scales from 1 to 8 GPUs; for larger graphs (bottom row), from 8 to 128 GPUs.

For smaller graphs, DistShap achieves a speedup of 4.6× to 7.3× compared to its runtime on a single GPU. Here, speedup depends on the single-GPU baseline runtime. For instance, Coauthor-Physics achieves a near-linear speedup of 7.3×, benefiting from a relatively high single-GPU runtime of around 200 seconds. In contrast, DBLP shows a 4.6× speedup since its single-GPU runtime is already quite low. For larger graphs, DistShap achieves a speedup of 11.1× to 13.5× when scaling from 8 to 128 GPUs. This near-linear speedup is due to the high baseline runtime on 8 GPUs for large datasets.

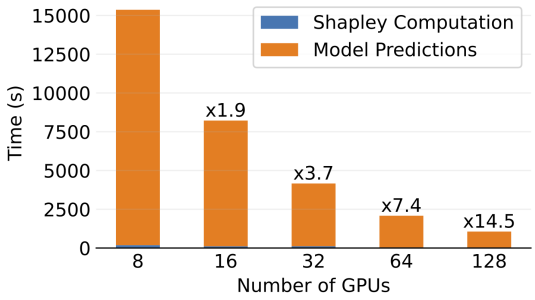

Figure 7: Scalability of DistShap on the Reddit dataset using a two-layer GAT model with 8 attention heads. Explanation time is measured for 50 nodes and 600K samples per node.

Table 4: Edges and memory required to store the largest 2-layer computational graphs and their features.

Graph	edges in the largest G_c	Memory
Coauthor-CS	3,299	0.04GB
Coauthor-Physics	18,622	0.15GB
DBLP	5,014	0.01GB
ogbn-arxiv	470,080	0.03GB
ogbn-products	2,722,095	0.11GB
Reddit	37,787,300	1.04GB

On small graphs, DistShap achieves 4.6×–7.3× speedups on 8 GPUs compared to its runtime on a single GPU. On large graphs, speedups range from 11.1×–13.5× on 128 GPUs compared to the runtime on 8 GPUs. In all cases, near-linear speedups are observed when the baseline runtime is relatively high, as seen in Coauthor-Physics, ogbn-arxiv, and Reddit.

Across all graphs, model prediction dominates runtime due to the large number of samples needed for better fidelity (e.g., 600K samples). Generally, model prediction time scales well because it does not require any communication among GPUs. In contrast, Shapley value computation is faster but scales less efficiently due to all-reduce overhead in the CGLS solver.

DistShap also scales well for other GNNs. Fig. 7 shows even better speedups with GAT, since GAT inference is more compute-intensive than GCN.

5.6 Ablation Studies

Memory overhead of replicated computational graph. In our implementation, we replicate the computational graph in every GPU before computing Shapley values. For most practical GNNs that use only two or three layers, their computational graphs fit within GPU memory, as shown in the table 4. However, deeper GNNs may expand a computational graph that spans a large portion of the input graph, leading to memory overflow. In such cases, memory usage can be managed using neighborhood sampling techniques used in GraphSAGE or GraphSAINT.

The impact of batching in model predictions. When explaining the prediction of a node, we batch 600K model predictions – one for each sampled subgraph – allowing each batch to leverage

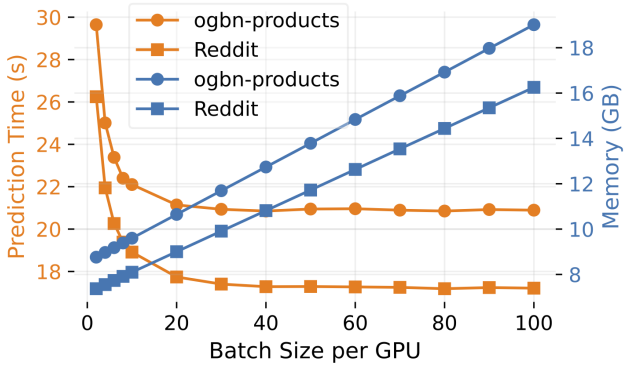

Figure 8: Model prediction times and memory consumption for different batch sizes. Node 18 in Reddit has 92,965 edges and node 236039 in ogbn-products has 97,019 edges in their respective computational graphs. 50,000 samples are used.

the parallelism available on each GPU. Fig. 8 shows that model prediction efficiency improves as the batch size increases, with optimal performance reached at a batch size of 50. Beyond this point, GPU utilization becomes saturated, and further increases in batch size yield no additional speedup. This performance gain, however, comes at the expense of higher memory consumption.

Communication time in Shapley calculations. We observed that communication accounts for approximately 2% of the total runtime of the CGLS solver during Shapley value computation. While communication does not scale as efficiently as computation, its overhead remains minimal – even when using 128 GPUs – and thus does not significantly affect the scalability of DistShap.

6 CONCLUSIONS

We present DistShap, a scalable GNN explanation method that uses distributed Shapley value computation. It improves over prior work by: (1) achieving high-fidelity explanations on large computational graphs via distributed sampling across GPUs, and (2) contributing efficient, scalable subgraph sampling, batched inference, and a GPU-parallel CGLS solver. Although we evaluate DistShap using GCN and GAT, the method is model-agnostic and applies to a broad range of GNNs with minimal modification, as it treats the model as a black box and relies solely on inference to assess edge contributions. For example, the approach naturally extends to heterogeneous and temporal graphs by treating edge types or time-stamped edges as players. The main challenge for such extensions lies in accommodating more complex graph data structures rather than in adapting the Shapley computation itself. Future work will focus on extending DistShap to other graph learning models.

ACKNOWLEDGMENTS

This research was funded in part by DOE grants DE-SC0022098 and DE-SC0023349 and by NSF grants CCF 2316233 and OAC-2339607. AD was supported by Oak Ridge National Laboratory through subcontract award number CW60249.

REFERENCES

[1] A. C. Aitken. 1936. IV.—On Least Squares and Linear Combination of Observations. *Proceedings of the Royal Society of Edinburgh* 55 (1936), 42–48.

[2] Selahattin Akkas and Ariful Azad. 2024. Gnnshap: Scalable and accurate gnn explanation using shapley values. In *Proceedings of the ACM Web Conference 2024*. 827–838.

[3] Selahattin Akkas and Ariful Azad. 2025. Shapley-Value-Based Graph Sparsification for GNN Inference. *arXiv preprint arXiv:2507.20460* (2025).

[4] Federico Baldassarre and Hossein Azizpour. 2019. Explainability techniques for graph convolutional networks. *arXiv preprint arXiv:1905.13686* (2019).

[5] Aleksandar Bojchevski and Stephan Günnemann. 2017. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815* (2017).

[6] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. 2012. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM Journal on Scientific Computing* 34, 1 (2012), A206–A239.

[7] Alexandre Duval and Fragkiskos D Malliaros. 2021. Graphsvx: Shapley value explanations for graph neural networks. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part II 21*. Springer, 302–318.

[8] Thorben Funke, Megha Khosla, Mandeep Rathee, and Avishek Anand. 2022. Zorro: Valid, sparse, and stable explanations in graph neural networks. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[9] Shurui Gui, Hao Yuan, Jie Wang, Qicheng Lao, Kang Li, and Shuiwang Ji. 2024. FlowX: Towards Explainable Graph Neural Networks via Message Flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 7 (2024), 4567–4578.

[10] Gérard Hamiache and Florian Navarro. 2020. Associated consistency, value and graphs. *International Journal of Game Theory* 49 (2020), 227–249.

[11] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the NeurIPS*. 1025–1035.

[12] Magnus R Hestenes, Eduard Stiefel, et al. 1952. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards* 49, 6 (1952), 409–436.

[13] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[14] Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, and Yi Chang. 2022. Graphlime: Local interpretable model explanations for graph neural networks. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[15] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[16] Wanyu Lin, Hao Lan, Hao Wang, and Baochun Li. 2022. Orphicx: A causality-inspired latent variable model for interpreting graph neural networks. In *Proceedings of the IEEE/CVF Conference on CVPR*. 13729–13738.

[17] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems* 30 (2017).

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

[19] Cristine Marsh and Isaac Joseph. 2020. Shparkley: Scaling Shapley Values with Spark. GitHub Repository. https://github.com/Affirm/shparkley.

[20] Andrea Mastropietro, Giuseppe Pasculli, Christian Feldmann, Raquel Rodríguez-Pérez, and Jürgen Bajorath. 2022. EdgeSHAPer: Bond-centric Shapley value-based explanation method for graph neural networks. *Iscience* 25, 10 (2022).

[21] Rohan Mitchell, Eibe Frank, and Geoff Holmes. 2022. GPUTreeShap: massively parallel exact calculation of SHAP scores for tree ensembles. *PeerJ Computer Science* 8 (2022), e880.

[22] Christopher C. Paige and Michael A. Saunders. 1982. Algorithm 583: LSQR: Sparse Linear Equations and Least Squares Problems. *ACM Trans. Math. Softw.* 8, 2 (June 1982), 195–209.

[23] Christopher C. Paige and Michael A. Saunders. 1982. LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares. *ACM Trans. Math. Softw.* 8, 1 (March 1982), 43–71.

[24] Tamara Pereira, Erik Nascimento, Lucas E Resck, Diego Mesquita, and Amauri Souza. 2023. Distill n'explain: explaining graph neural networks using simple surrogates. In *AISTATS*. PMLR, 6199–6214.

[25] Alan Perotti, Paolo Bajardi, Francesco Bonchi, and André Panisson. 2023. Explaining Identity-aware Graph Classifiers through the Language of Motifs. In *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[26] Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. 2019. Explainability methods for graph convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10772–10781.

[27] Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. 2021. Interpreting Graph Neural Networks for NLP With Differentiable Edge Masking. In *International Conference on Learning Representations*.

[28] Lloyd S Shapley. 1951. Notes on the n-person game—ii: The value of an n-person game. (1951).

[29] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).

[30] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*. 3319–3328.

[31] Juntao Tan, Shijie Geng, Zuohui Fu, Yingqiang Ge, Shuyuan Xu, Yunqi Li, and Yongfeng Zhang. 2022. Learning and evaluating graph neural network explanations based on counterfactual and factual reasoning. In *Proceedings of the ACM web conference 2022*. 1018–1027.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJXMpikCZ

[33] Minh Vu and My T Thai. 2020. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *Advances in neural information processing systems* 33 (2020), 12225–12235.

[34] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems* 32 (2019).

[35] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. 2020. Xgnn: Towards model-level explanations of graph neural networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 430–438.

[36] H. Yuan, H. Yu, S. Gui, and S. Ji. 2023. Explainability in Graph Neural Networks: A Taxonomic Survey. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 45, 05 (may 2023), 5782–5799.

[37] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. 2021. On explainability of graph neural networks via subgraph explorations. In *International conference on machine learning*. PMLR, 12241–12252.

[38] Shichang Zhang, Yozen Liu, Neil Shah, and Yizhou Sun. 2022. Gstarx: Explaining graph neural networks with structure-aware cooperative games. *Advances in Neural Information Processing Systems* 35 (2022), 19810–19823.

[39] Yue Zhang, David Defazio, and Arti Ramesh. 2021. Relex: A model-agnostic relational model explainer. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*. 1042–1049.

[40] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep learning on graphs: A survey. *IEEE TKDE* 34, 1 (2020), 249–270.