SliceGX: Layer-wise GNN Explanation with Model-slicing

Tingting Zhu
Zhejiang University
Zhejiang, China
tingtingzhu@zju.edu.cn

Tingyang Chen
Zhejiang University
Zhejiang, China
chenty@zju.edu.cn

Yinghui Wu
Case Western Reserve University
Cleveland, USA
yxw1650@case.edu

Arijit Khan
Aalborg University
Aalborg, Denmark
arijitk@cs.aau.dk

Xiangyu Ke
Zhejiang University
Zhejiang, China
xiangyu.ke@zju.edu.cn

Yunjun Gao
Zhejiang University
Zhejiang, China
gaoyj@zju.edu.cn

***Abstract*—Ensuring the trustworthiness of graph neural networks (GNNs) as black-box models requires effective explanation methods. Existing GNN explanations typically perform perturbations of model input and discover subgraphs that are responsible for the occurrence of the final output of GNNs. Such a scheme lacks the necessary finer granularity to perform internal, layer-wise explainability analysis. The need for such ability is evident in model diagnosis and architecture optimization. This paper introduces SliceGX, a novel GNN explanation approach that generates explanations at specific GNN layers in a progressive manner. Given a GNN $\mathcal{M}$, selected intermediate layers, and a target layer, SliceGX automatically segments $\mathcal{M}$ into layer blocks ("model slice") and discovers high-quality explanatory subgraphs at each layer block that clarifies the occurrence of output of $\mathcal{M}$ at the targeted layer. While this problem is computationally hard, we develop efficient algorithms and optimization techniques to generate and maintain such explanations incrementally. Through experiments on real-world graphs and representative GNN classes, we verify the effectiveness and efficiency of SliceGX, and showcase how it supports model debugging.**

***Index Terms*—Graph neural networks, Explainable AI**

I. Introduction

Graph Neural Networks (GNNs) have demonstrated promising performances in various applications. A GNN is a function $\mathcal{M}$ that converts an input graph G to a numerical matrix representation ("graph embedding") Z, which can be further post-processed to output task-specific answers, such as node classification and graph classification.

Despite their expressive capabilities, ensuring the trustworthiness and reliability of graph learning models requires more than just accurate predictions—it calls for transparent tracking of the inference process [4, 63]. Understanding how a GNN arrives at a decision is essential for model provenance and tracking [37]. For example, a provenance-oriented analysis [4] may ask "*What*" questions that explain what happens with relevant features, "*How*" they are processed by the subsequent layers of the model, or "*Where/When*" the contribution of different layers are in a neural network. Such finer-grained, "layer-wise" interpretation, as also practiced in other deep models, e.g., in CNNs [27] and DNNs [33, 48], further benefits model optimization and debugging [2, 13].

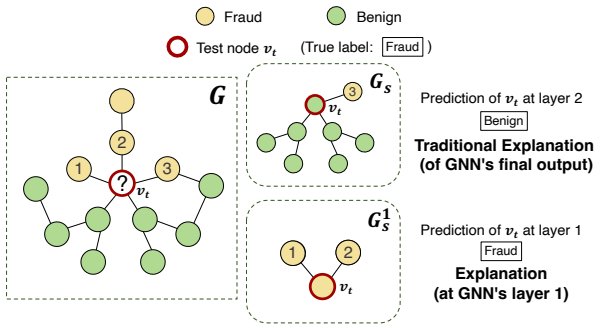


Fig. 1. Generating Layer-wise Explanations for GNN diagnosing for spam review detection [15].

Despite the evidential need, the study of layer-wise interpretation for GNNs, and how it benefits GNN applications, are still in its infancy. Consider the following example.

Example 1. *A fraction of a spam review network G [15] is illustrated in Fig. 1, where nodes represent reviews and edges denote associations between reviews based on shared reviewers, common products, or temporal links. A GNN model classifies reviews as either "fraud" (yellow) or "benign" (green), with fraud reviews exhibiting deceptive or spam-like behavior.*

Observing that a fraud review v_t is incorrectly labeled as "benign", the analyst may want to diagnose the inference process of the GNN $\mathcal{M}$. Several "diagnose" queries, analogous to those in CNNs and DNNs [27, 41, 48], can be posed:

- *"**Q1**:* At which layer (When) did $\mathcal{M}$ first make this mistake*?" This helps the analyst to "stop on the first error" at prior layers for early intervention;*
- *"**Q2**:* Which fraction of G (subgraphs) are responsible for this first mistake*?" This identifies influencing nodes, links, and their features that lead to the above mistake;*
- *"**Q3**:* How did the influence propagate via subsequent

layers*?" This reveals the contributions of these nodes across layers.*

A closer look at the inference process of $\mathcal{M}$ informs us the following information.

*(1) For **Q1**, by treating each layer as an "output layer", we can "segment" the inference process and analyze the node embedding of v_t at a specific layer to assess if the accumulated neighborhood impact leads to an incorrect label. The established strategy that aggregates localized neighbors such as "soft labels" [53] to track node influence also justifies this intuition. For example, v_t is correctly classified as "fraud" at layer 1 but incorrectly labeled as "benign" at layer 2, alerting a first error at layer 2 for further investigation.*

*(2) For **Q2**, existing GNN explainers [22, 25, 45, 54, 56] such as GNNExplainer [54] (see Related Work) provide factual or counterfactual subgraph explanations that are "faithful" to the label. However, they typically only explain the final output and struggle to adapt to an intermediate layer as derived for **Q1**. Moreover, the contributions of nodes and edges can vary across layers. Indeed, the subgraphs G_s^1 and G_s at layer 1 and 2 (in Fig. 1) highlight different influences and feature structures responsible for its local "fraud" and "benign" predictions, respectively.*

*(3) To answer **Q3**, one needs to also necessarily annotate the explanation structures by distinguishing between influencing nodes and edges that contribute most to the output at targeted layers and those that are only "connecting" and propagating such influence during the inference. For example, the graph G_s may indicate that the first mistake is due to accumulated influence from a majority of "benign" 2-hop neighbors of v_t at layer 2, while an adjacent edge of v_t merely propagates this influence, hence suggesting further investigation for the "How" question **Q3**. Traditional GNN explainers do not bear such annotated expressiveness.*

The above example calls for a layer-wise explanation framework for GNNs. Such a framework should efficiently generate layer-wise explanations that are consistent and concise for the results at intermediate layers of interests during GNN inference. We aim to address the following questions: (1) *How to define explanation structures for layer-wise GNN explanations?* (2) *How to measure the explainability of layer-wise explanations?* and (3) *How to efficiently generate layer-wise GNN explanations?*. None of the current GNN explainability methods fully address these questions.

Contributions. Our main contributions are as follows.

(1) We, for the first time, extend conventional GNN explanation to its layer-wise counterpart. (a) We introduce a class of explanatory subgraphs that specify a set of explanatory nodes and connection edges, to enable layer-wise influence analysis. (b) We also introduce "model slice", inspired and obtained by model slicing, which allows us to create a sub-model by stacking user-designated source layers and a target layer to characterize and conduct layer-wise GNN explanation.

(2) We present SliceGX, a layer-wise, novel GNN explanation framework to discover explanatory subgraphs at designated layers. We define layer-wise GNN explanations as factual or counterfactual explanations at specific layers. To guide layer-wise GNN generation, we propose a novel bi-criteria quality measure that integrates message-passing influence and embedding diversity, enabling precise layer-specific explanations for designated nodes of interest. We also establish the hardness of explanation generation under model slicing.

(3) We present novel and efficient post-hoc explanation generation algorithms to discover connected subgraph explanations from specific model layers without retraining. Leveraging GNNs' inherent data locality, our method employs a greedy node selection strategy optimized for high explainability. We present an efficient explanation generation algorithm for single-node single source layer, specify $\frac{1}{2}$-approximation guarantee, and further extend it to multiple nodes and multiple source layers. We also integrate SPARQL-like query support, allowing users to interactively explore GNN behavior at each layer without writing detailed debugging code.

(4) We verify the performance of SliceGX with extensive experiments on benchmark datasets and explanation tasks. Our results demonstrate the framework's efficiency, the effectiveness of generated explanations, and its scalability through parallelization in large-scale graphs (with hundreds of millions of edges). Our framework also reveals new insights into GNN behavior, enabling downstream applications, i.e., model debugging and model optimization.

II. Related Work

Explainability of GNNs. Several approaches have been proposed to explain GNNs' predictions [22, 25, 45, 54, 56], primarily by identifying important nodes, edges, node features, subgraphs, and message flows. An intuitive way is to monitor the prediction changes with respect to input perturbations [55]. For example, GNNExplainer [54] learns soft masks for edges and node features to maximize the mutual information between the original and perturbed graph predictions. SubgraphX [56] efficiently samples the subgraphs from an input graph by employing pruning-based Monte Carlo Tree Search (MCTS) and Shapley Value. GraphMask [45] trains a classifier to predict whether an edge can be dropped from each GNN layer without affecting the original predictions. FlowX [22] quantifies the importance of message flows based on Shapley values, identifying critical propagation pathways. Previous work has also studied the surrogate-based methods, which employ a model to approximate the predictions of GNN [16, 25, 39, 61].

Nevertheless, these methods cannot be directly applied to find the layer-wise explanation for GNNs. (1) They are limited to identifying the most influential components contributing to the prediction for the final output of GNNs, hence lacking finer-grained analysis capacity. For example, they are not sufficiently expressive to yield a sequence of explanations to reveal how a final explanation is formed in a progressive, layer-by-layer manner. (2) They fail to distinguish between the roles of influential nodes and the structures connecting them. (3)

They do not provide an explanation accessing mechanism to allow a customizable explanation generation process, limiting their applicability in practice.

Layer-wise GNN Analysis. Several methods are proposed for layer-wise GNNs analysis. Decomposition-based methods, i.e., GNN-LRP [46] and Excitation-BP [42], require node weights per layer to compute input importance scores. GraphMask [45] generates edge masks for each layer but still focuses on explaining the final output rather than specific target layers. KerGnns [17] provides self-interpretability with hierarchical graph filters, generating explanations tailored to specific GNN architectures.

Declarative Explanation Queries. Declarative explanation queries allow users to specify what explanation they need without detailing the computational method to achieve it [5]. Several works [9, 26] have prompted the proposal for declarative explainability languages to provide users with the flexibility to interact with an ML model by posing various queries in pursuit of the optimal explanation. For GNNs, PL4XGL [28] designs a declarative programming language to explain classification results. However, this language serves as a framework for articulating explanations and is not designed to invoke the explanation process. In our work, instead of developing a full-fledged symbolic explanation language, we provide convenient shorthand wrapped operators that can be incorporated into a fragment of SPARQL-like statements and focus on how to encourage users to utilize these operators to precisely manipulate declarative explanations.

Model Slicing. Model slicing, inspired by program slicing [50], involves extracting sub-models for finer-grained analysis [7], such as efficient inference [8, 58], adversarial detection [62], mechanistic interpretability [44], and visual analytics [24]. This paper makes a first step to exploit model slicing to enable layer-wise GNN explanation. We envision SliceGX to scale GNN debugging and interpretation, as verified by our case analysis.

III. Graph Neural Networks and Explanation

Graphs. A graph $G = (V, E)$ has a set of nodes V and a set of edges E. Each node v carries a tuple, representing its attributes (or features) and their corresponding values.

We define the following terms: (1) The l-hop neighbors of a node $v \in V$, denoted $N^l(v)$, are nodes *within* l hops from v in G. (2) The l-hop neighbor subgraph, $G^l(v)$, is a node induced subgraph of G, which is induced by $N^l(v)$. (3) For a set of nodes $V_s \subseteq V$, the l-hop neighbor subgraph, $G^l(V_s)$, is induced by the union of l-hop neighbors of nodes in V_s, i.e., $\bigcup_{v \in V_s} N^l(v)$. $G^l(v)$ is always connected, while $G^l(V_s)$ may not be. (4) The size of a graph is the total number of nodes and edges, given by $|G| = |V| + |E|$.

Graph Neural Networks. GNNs are a class of neural networks that learn to convert graphs to proper representations. Their output can be post-processed for tasks such as node classification. We provide an abstract characterization of a GNN $\mathcal{M}$ with L layers.

In general, a GNN can be considered as a composition of two functions: an encoder f_1 and a predictor f_2.

(1) The encoder f_1 is a composite function $f_1^L \circ f_1^{L-1} \circ \ldots \circ f_1^1(G)$. At the l-th layer ($l \in [1, L]$), the function f_1^l uniformly updates, for each node $v \in V$, an input representation from the last layer as:

$$z_v^l := \mathsf{TRANS}(\mathsf{AGG}^l(z_j^{l-1} | j \in N^l(v) \cup \{v\}) \qquad (1)$$

Here, z_v^l is the feature representation ("embedding") of node v at the l-th layer (z_v^0 is the input feature of v); N_v is the (1-hop) neighbors of v in G, and AGG^l (resp. $TRANS^l$) represents an aggregation (resp. transformation) function at the l-th layer, respectively.

(2) Subsequently, a predictor f_2 takes the output embeddings $f_1(G) = Z$ and performs a task-specific post-processing $f_2(Z)$, to obtain a final output (a matrix), denoted as $\mathcal{M}(G)$. Here f_2 typically includes a multi-layer perceptron (MLP) that takes the aggregated representations Z_v from the encoder f_1, and produces the final output for the specific task, such as node classification or graph classification. Specifically, given a node $v \in V$, the output of $\mathcal{M}$ at v, denoted as $\mathcal{M}(G, v)$, refers to the entry of $f_2(Z)$ (i.e., $\mathcal{M}(G)$) at node v.

The above step is an abstraction of the inference process of $\mathcal{M}$. Variants of GNNs specify different aggregation and transformation functions to optimize task-specific output. Notable examples include Graph Convolutional Networks (GCNs) [32], a representative class of GNNs, which approximate the first-order Eigen decomposition of the graph Laplacian to iteratively aggregate neighbor embeddings and specify TRANS as nonlinear activations, e.g., ReLU.

Consider the inference process for node classification, a cornerstone task in graph analysis. The task of node classification is to assign a categorical class label for a node. A GCN-based node classifier $\mathcal{M}$ learns to convert an input graph G with Z^0 (a feature matrix) and a (normalized) adjacency matrix to an output representation Z. The inference process of $\mathcal{M}$ applies Equation 1 to generate output Z with the encoder function f_1. The post processing f_2 converts Z to map each node embedding $z_{v_t} \in Z$ to a node label, for each node v_t in a designated test node set $V_T \subseteq V$.

Explanation Structure of GNNs. We start with a notion of explanation. Given a GNN $\mathcal{M}$ with L layers, a graph $G = (V, E)$, and an output $\mathcal{M}(G, v_t)$ to be explained, an *explanation* of $\mathcal{M}(G, v_t)$ is a node-induced, connected subgraph $G_s = (V_s \cup V_c, E_s)$ of G, where

- (1) $V_s \subseteq V$ is a set of *explanatory nodes* that clarify the output $\mathcal{M}(G, v_t)$ ($v_t \in V_s$) [1],
- (2) $V_c \subseteq V$ is a *connector set* induced by explanatory nodes V_s, which is defined as $\bigcup_{v_s \in V_s} C(v_s)$ with $0 \leq |V_c| < |N^L(v_t)|$; here $C(v_s)$ is the set of nodes on the paths from v_s to v_t within $N^L(v_t)$;

[1]We introduce explanatory nodes as a "role" in explanatory structures; we will specify how to discover explanatory nodes for layer-wise explanation in Sections IV and V.

- In addition, *one* of the following conditions holds: **(1)** $\mathcal{M}(G, v_t) = \mathcal{M}(G_s, v_t)$, i.e., G_s is a "factual" explanation; or **(2)** $\mathcal{M}(G, v_t) \neq \mathcal{M}(G \setminus G_s, v_t)$, i.e., G_s is "counterfactual".

Here, $G \setminus G_s$ is a graph obtained by removing the edge set E_s from G, while retaining all nodes. The inference process of $\mathcal{M}$ over G_s (or $G \setminus G_s$) determines if G_s is factual (or counterfactual). The input for inference is always a feature matrix Z^0 and an adjacency matrix, regardless of graph connectivity.

Remarks. The above definition is well justified by established GNN explainers, which generate (connected) subgraphs that are factual [54, 56], counterfactual [6, 35], or both [11], as variants of the above definition. The main difference is that we explicitly distinguish explanatory nodes and connect sets. The added expressiveness allows us to track the varying set of "influencing" nodes and edges at different layers (as will be discussed in Section IV-A). This is not fully addressed by prior GNN explainers.

The rest of the paper makes a case for GCN-based node classification. For simplicity, we refer to "class labels" as "labels", and "GCN-based classifiers" as "GCN". Our approach *applies to other representative* GNNs in Appendix C.1, as verified in our evaluations.

IV. SliceGX: Layer-wise Explanations

A. Extending Explanations with Model-Slicing

We start with a notation of the model slice.

Model Slices. To gain insights into the inner-working of GNNs and to enable a more granular interpretation, inspired by model slices [7], we create "model slices" of a trained GNN $\mathcal{M}$ with (user-selected) individual layers.

Given a GNN $\mathcal{M}$ as a stack of an L-layered encoder f_1 and a predictor f_2, and a selected layer $l \in [1, L]$, an *l-sliced model* of $\mathcal{M}$, denoted as $\mathcal{M}^l$, is a GNN as a stack of an encoder f^l and the predictor f_2, where f^l is a composite function in the form of.

$$Z^l = f_1^l \circ f_1^{l-1} \circ \ldots \circ f_1^1(G) \tag{2}$$

The inference process of an l-sliced model $\mathcal{M}^l$ computes a result consistently by (1) calling a composition $f_1^l \circ \ldots f_1^1(G)$ to generate output embeddings Z^l up to layer l, and (2) deriving $z_{v_t}^l \in Z^l$ and applying $f_2(z_{v_t}^l)$ to obtain the result $\mathcal{M}^l(G, v_t)$, for each node $v_t \in V_T$. In our abstraction, every function f_1^l ($l \in [1, L]$) and f_2 uniformly takes a $|V| \times d$ embedding matrix as input and produces a matrix of the same size. In practice, we assume that the node features' dimensionality d remains unchanged across layers.

Example 2. *Fig. 2 (upper part) depicts the architecture of an l-sliced model $\mathcal{M}^l$. Given a* GNN *$\mathcal{M}$ with L layers, the encoder in derived $\mathcal{M}^l$ includes the first l* GNN *layers $f_1^1, ..., f_1^l$ for output embedding generation. The predictor f_2 in $\mathcal{M}^l$ remains the same as in $\mathcal{M}$. The model parameters in the corresponding modules are consistent across $\mathcal{M}$ and $\mathcal{M}^l$.*

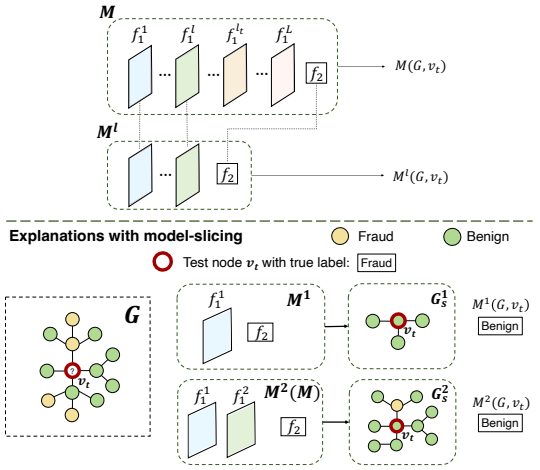


Fig. 2. Example of explanations with model-slicing. The upper part depicts the architecture of an l-sliced model $\mathcal{M}^l$, and the lower part shows factual explanations at each layer of $\mathcal{M}$ for diagnosing purposes as a test node v_t is misclassified with the label "Benign" by $\mathcal{M}$. G_s^l is the explanation at layer l that clarifies "why" the inference gets a wrong prediction.

Explaining GNNs **with Model Slicing**. We naturally extend GNN explanations with model slicing. Given a node $v_t \in V_T$ with an output $\mathcal{M}(G, v_t)$ to be explained, a connected subgraph G_s^l of G is an *explanation* of $\mathcal{M}(G, v_t)$ *at layer l*, if one of the following holds:

(1) The subgraph G_s^l is a factual explanation of $\mathcal{M}(G, v_t)$, that is, $\mathcal{M}(G, v_t) = \mathcal{M}(G_s^l, v_t)$; and also (b) $\mathcal{M}(G, v_t) = \mathcal{M}^l(G, v_t) = \mathcal{M}^l(G_s^l, v_t)$. In this case, we say that G_s^l is a *factual explanation of $\mathcal{M}(G, v_t)$ at layer l*; or

(2) The subgraph G_s^l is a counterfactual explanation of $\mathcal{M}(G, v_t)$, that is, $\mathcal{M}(G, v_t) \neq \mathcal{M}(G \setminus G_s^l, v_t)$; and also (b) $\mathcal{M}(G, v_t) = \mathcal{M}^l(G, v_t) \neq \mathcal{M}^l(G \setminus G_s^l, v_t)$. In this case, we say that G_s^l is a *counterfactual explanation of $\mathcal{M}(G, v_t)$ at layer l*.

Intuitively, we are interested at finding an explanation for an output $\mathcal{M}(G, v_t)$, which can be consistently derived by a sliced model at layer l, an "earlier" stage of inference of GNN $\mathcal{M}$. We also provide a detailed discussion and additional experimental results in Appendix A.1, where we show that this approach yields faithful explanations that align with the model's true decision-making process.

Example 3. *Consider a two-layer* GNN*-based node classifier $\mathcal{M}$ as in Example 2 (lower part of Fig. 2). The test node v_t in graph $G = (V, E)$ is misclassified as "Benign" by $\mathcal{M}$, and its two "sliced" counterparts $\mathcal{M}^1$, and $\mathcal{M}^2$, at layer 1 and layer 2, respectively. Here, G_s^1 and G_s^2 are factual explanations for the incorrect prediction at layers 1 and 2, respectively. While G_s^2 ensures consistency between $\mathcal{M}(G, v_t)$ and $\mathcal{M}(G_s^2, v_t)$ at the final layer, G_s^1 must maintain consistency in predictions for v_t in both $\mathcal{M}$ and the sliced model $\mathcal{M}^1$. Specifically, $\mathcal{M}^1$ assigns v_t the label "Benign" with G_s^1, aligning with $\mathcal{M}$'s final prediction. Inspecting the two explanations, G_s^1 and G_s^2, one can gain insights by observing the specific fraction of*

G that may be responsible for an incorrect label "Benign" of node v_t. Better still, the explanations G_s^1 and G_s^2, putting together, suggest a "chain of evidence" that clarifies the misclassification of v_t by $\mathcal{M}$ at early stage (layer 1) and its subsequent layers.

B. Explainability Measure

While factual and counterfactual explanations are adopted to characterize "relevant data" that clarify GNN outputs, it is necessary to quantify their explainability for specific diagnosing and interpreting needs. We adopt a novel bi-criteria measure below, in terms of label influence and embedding diversity, each with justification.

Bi-criteria Explainability Measure. Given a graph $G = (V, E)$, a GNN $\mathcal{M}$ with L layers, a specific layer number l ($l \in [1, L]$), and a designated node v_t for which the output $\mathcal{M}(G, v_t)$ needs to be explained, the *explainability* of an explanation subgraph G_s^l, with the explanatory node set V_s^l, at layer l is defined as:

$$f(G_s^l) = \gamma I(V_s^l) + (1 - \gamma) D(V_s^l) \tag{3}$$

where (1) $I(V_s^l)$ quantifies the relative influence of V_s^l; (2) $D(G_s^l)$ measures the embedding diversity of V_s^l; and (3) $\gamma \in [0, 1]$ is a balance factor between them. Note that $f(G_s^l) \in [0, 1]$.

We next introduce the functions $I(\cdot)$ and $D(\cdot)$.

Relative influence. We extend GNN feature propagation [11, 59] to layer-wise analysis. We introduce *relative influence*, a measure to evaluate the influence of a node v at a specific layer to a node u with output $\mathcal{M}(G, u)$. The relative influence of a node v at layer l on u is quantified by the L1-norm of the expected Jacobian matrix:

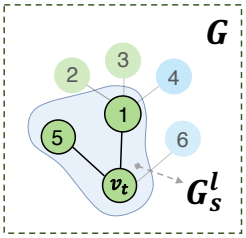
$$I_1(v, u) = \left|\left|\mathbf{E}[\frac{\partial Z_v^l}{\partial Z_u^0}]\right|\right|_1 \tag{4}$$

Further, we define the normalized relative influence as $I_2(v, u) = \frac{I_1(v,u)}{\sum_{w \in V} I_1(w,u)}$. We say that the node u is *influenced* by v at layer l, if $I_2(v, u) \geq h$, where h is a threshold for influence. The Jacobian matrix can be approximately computed in PTIME [12, 52].

Given an explanation G_s^l with node set V_s^l, and $N^l(v)$ as l-hop neighbors of $v \in V_s^l$, the *relative influence set* of v is $c(v) = \{u | I_2(v, u) \geq h, u \in N^l(v)\}$. Intuitively, it refers to the nodes that can be "influenced" by a node v in the explanation G_s^l; the larger, the more influencing node v is.

We next introduce a node set function $I(V_s^l) = \frac{1}{|V|}\left|\bigcup_{v \in V_s^l} c(v)\right|$. Given an explanation G_s^l, it measures accumulated influence as the union of $c(v)$ for all $v \in V_s^l$. The larger, the more influential G_s^l is.

Embedding diversity. Optimizing layer-wise explanations in terms of influence analysis alone may still lead to biased, "one-sided" explanations that tend to be originated from the similar type of most influential nodes across layers. We thus consider diversification of explanation generation to mitigate such bias, and enrich feature space [60].



Node	Influence set	Diversity set
v_t	{1,5,6}	{5}
1	{2,3,4, v_t}	{5,6}
2	{1,v_t}	{5}
3	{1}	{5}
4	{1}	{}
5	{v_t}	{1,2,3,v_t}
6	{v_t}	{1}

Fig. 3. Explanation with high influence and high embedding diversity. For simplicity, we consider a connected, node-induced explanation consisting only of explanatory set.

Given an explanation G_s^l at layer l of an output $\mathcal{M}^l(G, u)$ of a node u, we define the *diversity set* $r(v)$ as the nodes in $N^l(v)$ with a distance score d above threshold θ, i.e., $r(v) = \{u | d(Z_v^l, Z_u^l) \geq \theta, u \in N^l(v)\}$. The distance function d can be Euclidean distance of node embeddings [34], and $D(V_s^l)$ quantifies diversity as the union of $r(v)$ for all $v \in V_s^l$: $D(V_s^l) = \frac{1}{|V|}\left|\bigcup_{v \in V_s^l} r(v)\right|$.

Putting these together, we favor explanation with better explainability, which can contribute (1) higher accumulated influence relative to the result of interests from current layers, and (2) meanwhile reveals more diverse nodes on its own, in terms of node embedding. Our method is configurable: other influence and diversity (e.g., mutual information [54]) or explanability measures (e.g., fidelity [19]) can be used to generate explanations for task-specific needs.

Example 4. *Fig. 3 showcases an explanation with high explainability. For simplicity, we consider a connected, node-induced explanation subgraph consisting only of explanatory set. The influence set and diversity set of the explanation G_s^l with node set $V_s^l = \{1, 5, v_t\}$ is $\{1, 2, 3, 4, 5, 6, v_t\}$ and $\{1, 2, 3, 5, 6, v_t\}$, respectively. Among all the possible explanations of G, G_s is is favored with both maximum high relative influence and high embedding diversity, resulting in the maximum quality score calculated in Equation 3.*

C. Explanation Generation Problem

Configuration. We next formulate the problem of explanation generation with model slicing. We characterize the explanation request as a *configuration* $\mathcal{C} = (G, \mathcal{M}, v_t, \text{Ł}, l_t, k)$, which specifies:

- a graph G, a GNN M with L layers, where the l-th layer is indexed with a layer number l;
- a *target layer* l_t and a *target node* v_t, to specify a designated output $\mathcal{M}^{l_t}(G, v_t)$ to be explained;
- *source layers* Ł, a set of user-specified layer numbers, such that for each layer $l \in$ Ł ($l \in [1, L]$), an l-sliced model $\mathcal{M}^l$ is derived to provide explanations for $\mathcal{M}^{l_t}(G, v_t)$;
- k is a size bound w.r.t. the number of explanatory nodes.

Given a target result $\mathcal{M}^{l_t}(G, v_t)$ to be explained, the explanation generation with model-slicing is to generate, at each layer $l \in$ Ł, an optimal explanation G_s^{*l} of G with an explanatory node set V_s, for the output $\mathcal{M}^{l_t}(G, v_t)$, such that

$$G_s^{*l} = \underset{|V_s| \leq k}{\arg\max} f(G_s^l) \tag{5}$$

The configuration supports progressive analysis of GNN outputs: (1) For "progressive" diagnosis, answering why $\mathcal{M}$ assigns a wrong label $\mathcal{M}(G, v_t)$ (Example 2), set $\mathcal{C}$ as $(G, \mathcal{M}, v_t, \{1, \ldots, L\}, L, k)$, where L is the output layer. (2) To analyze $\mathcal{M}$'s behavior at selected layers for target layer l_t, set $\mathcal{C}$ as $(G, \mathcal{M}, v_t, Ł, l_t, k)$. (3) Conventional GNN explanation is a special case with $Ł = \{L\}$ and $l_t = L$.

Complexity. We next study the hardness of the problem. We first show that it is feasible to check if a subgraph is an explanation.

Lemma 1. *Given a configuration $\mathcal{C} = (G, \mathcal{M}, v_t, Ł, l_t, k)$, and a subgraph G_s of G, it is in* PTIME *to verify if G_s is an explanation w.r.t. $\mathcal{M}^{l_t}(G, v_t)$ at layer l.*

Proof sketch: We consider a configuration C for a fixed GNN model $\mathcal{M}^{l_t}$ at layer l_t. As shown in cost analyses for GNNs, $\mathcal{M}^{l_t}$ is constant with the same architecture and weights, ensuring deterministic inference in PTIME. To verify G_s, we evaluate a Boolean query for "factual" or "counterfactual" conditions at layer l_t, involving limited calls to $\mathcal{M}$ and equality tests, like $\mathcal{M}(G, v_t) = \mathcal{M}^{l_t}(G_s, v_t)$, all within PTIME.

Theorem 1. *The decision problem of Explanation generation with model slicing is* **NP***-hard.*

Proof: We perform a reduction from the maximum coverage problem [31]. Given a collection of sets $\mathcal{S} = S_1, \ldots S_n$, and an integer k, the maximum coverage problem is to find a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \leq k$, and the number of elements $|\bigcup_{S \in \mathcal{S}'} S|$ is maximized (or $\geq b$ for a given threshold b, for its decision version).

Given k, $\mathcal{S}'$, and $\mathcal{S}$, we construct a configuration as follows. (1) We construct a graph G with $V = \{v_t\} \cup V_1 \cup V_2$, where v_t is a designated target node; for each set $S_i \in \mathcal{S}$, there is a node v_{S_i} in V_1; and for each element $x_j \in S_i \in \mathcal{S}$, there is a distinct node v_j in V_2. (2) For every element $x_j \in S_i$, there is an edge (v_j, v_{S_i}) in E. For every node v_{S_i}, there is an edge between v_{S_i} and v_t. (3) Assign all the nodes in G a same ground truth label. Train a 2-layer vanilla GNN $\mathcal{M}$ that assigns a correct label to v_t (which is in PTIME [10]). We set $L = 2$, $Ł = \{1\}$, and $l_t = 2$, i.e., to request an explanation at layer 1 for a final result $\mathcal{M}(G, v_t)$ at output layer 2, with size at most k. We use G as a test graph. We set $\gamma = 1$ to favor the nodes with maximized influence, and a threshold b for explainability $f(\cdot)$.

As $\mathcal{M}$ is a fixed model that takes as the same input G as both training and test data, the inference ensures the invariance property of GNNs [20] and assigns $\mathcal{M}(G, v)$ with ground truth label for every $v \in V$. For each node v_{S_i}, the relative influence set $c(v_{S_i})$ is exactly the set of nodes $\{v_{ij}\}$, which corresponds to the set S_i and its elements. Then there exists a size k factual explanation of v_t (which is a connected star that contains v_t and up to k of its direct neighbors, for sliced model $\mathcal{M}^1$) with explainability at least b, if and only if there is a solution for maximum coverage with $|\bigcup_{S \in \mathcal{S}'} S| \geq b$. As the maximum coverage problem is known to be NP-hard [31], our problem is **NP**-hard.

Despite the hardness, we introduce novel and efficient algorithms for level-wise explanation generation, with guarantees on the quality.

Algorithm 1 : Slice(SS) (single node, single source layer)

Input: a configuration $\mathcal{C} = (G, \mathcal{M}, v_t, \{l\}, l_t, k)$;
Output: an explanation G_s^l w.r.t. $\mathcal{M}^{l_t}(G, v_t)$ at GNN layer l.

1: initialize $G_{v_t}^l = (V_{v_t}^l, E_{v_t}^l)$ as the l-hop subgraph in G
2: set $V_s := \emptyset$; $V_c := \emptyset$
3: **while** $|V_s| < k$ and $V_{v_t}^l \backslash V_s \neq \emptyset$ **do**
4: expand V_s by pair-wise node selection in $V_{v_t}^l$
5: select $V_c \in V_{v_t}^l$ to keep induced subgraph connected
6: $V_s' = V_s \cup V_c$
7: set G_s^l as the subgraph derived from node set V_s';
8: **if** Verify (G_s^l, C) = true **then**
9: **return** G_s^l;
10: **while** Verify(SS) (G_s^l, C) = false and $V_{v_t}^l \backslash V_s' \neq \emptyset$ **do**
11: replace the lowest-scoring node in V_s with highest-priority node in $V_{v_t}^l \backslash V_s'$ while ensuring connectivity
12: set G_s^l as the subgraph derived from node set V_s';
13: **if** Verify (G_s^l, C) = true **then**
14: **return** G_s^l;
15: **return** $G_{v_t}^l$.

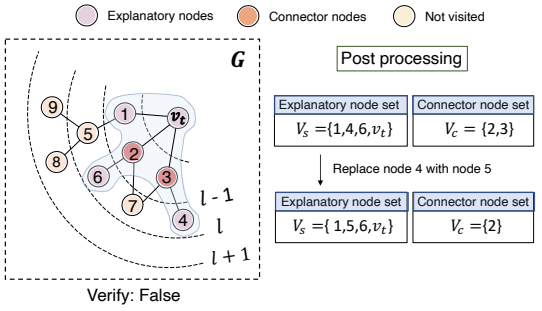


Fig. 4. A running example of SliceGX (k=4)

V. Generating Model-Slicing Explanation

Our first algorithm generates layer-wise explanations for a single node at a designated target layer.

A. Single-Node, Single-Source Layer

Algorithm. The algorithm, denoted as Slice(SS) and illustrated as Algorithm 1, takes a configuration $\mathcal{C}$ as input, and generates an explanation G_s^l for the output $\mathcal{M}^{l_t}(G, v_t)$ of a single test node v_t, at a specific layer l.

Initialization (lines 1-2). Algorithm Slice(SS) leverages data locality in GNN inference: For node v_t, an l-hop GNN computes $\mathcal{M}(G, v_t)$ by visiting l-hop neighbors via message passing. Prior work treats the l-hop subgraph as the computational graph, where l is the number of layers [36]. For layer l ($l \in [1, L]$), Slice(SS) initializes $G_{v_t}^l$ as the l-hop subgraph

centered at test node v_t, induced by nodes and edges within l-hop of v_t, to refine the search space. It also computes node representations and distance scores for $G^l_{v_t}$.

Generation Phase (lines 3-9). Slice(SS) then applies a greedy node selection strategy to find a set of explanatory nodes V_s, and then induces the explanation by definition. The greedy selection process iteratively chooses a pair of nodes that maximize a marginal gain of $f(\cdot)$ (line 4). It then induces an explanation with the connect sets of each explanatory node V_s, by definition (Section III). It finally invokes a procedure Verify(SS) to check if G^l_s satisfies constraints to be an explanation, by definition, and return G^l_s if so.

Post-processing (lines 10-15). The generation phase does not guarantee that G^l_s passes Verify. Thus, a lightweight post-processing phase is added. "Back up" nodes are sorted by explainability scores, replacing the lowest-scoring node in V_s while preserving connectivity. The Verify procedure checks if the subgraph yields factual or counterfactual explanations and returns it if valid.

Finally, if no explanation is identified, Slice(SS) returns $G^l_{v_t}$, the l-hop subgraph of v_t, a factual explanation of $\mathcal{M}(G, v_t)$ by default.

Procedure Verify. The procedure verifies if the subgraph G^l_s induced by $V_s \cup V_c$ is a valid explanation, by determining if it is a factual or counterfactual explanation at layer l. In addition, it generates necessary annotation information to support declarative access with query semantics, as will be discussed in Section V-D. Following Lemma 1, the verification is in PTIME.

Example 5. *Fig. 4 illustrates an explanation generated by* Slice(SS). *The explanatory node set $V_s = \{1, 4, 6, v_t\}$ is initially chosen to maximize the explainability score with the desired size k. To ensure connectivity, connector nodes $V_c = \{2, 3\}$ are added, but the subgraph fails verification. So, the node 4 is then replaced with node 5, updating V_c to $\{2\}$, resulting in a valid explanation with $V_s = \{1, 5, 6, v_t\}$ and $V_c = \{2\}$.*

Remarks. In practice, the post-processing phase in Algorithm 1 (line 10–15) is triggered in only around 3% of cases to enforce this condition. After post-processing, valid explanations fail to emerge in merely 1% of instances. In these rare cases, the algorithm conservatively returns $G^l_{v_t}$ as the explanation (Algorithm 1, line 15)—the only scenario where the condition of the explanations in Section IV-A may not be fully satisfied. This highlights the robustness of our method, which successfully produces faithful explanations that align with the model's behavior in the vast majority of cases.

Correctness and Approximability Algorithm Slice(SS) ensures correctness by selecting V_s and inducing their connect set, ensuring G^l_s is either a factual or counterfactual explanation at layer l. For approximability, we present the following detailed analysis:

Lemma 2. *Given an output $\mathcal{M}(G, v_t)$ and an explanation G^l_s at layer l, the explainability measure $f(G^l_s)$ is a monotone submodular function for the node set of G^l_s.*

Proof: As $f(G^l_s)$ is the sum of two node set functions $I(V^l_s)$ and $D(V^l_s)$ defined in Equation 3, we prove that the two functions are both monotone submodular. We first analyze the property of $I(V^l_s)$. **(1)** It is clear that the relative influence set $c(v)$ for any $v \in V^l_s$ is non-negative. So $I(V^l_s)$ is non-decreasing monotone function. **(2)** Next we will show that for any set $V_{s'} \subset V_s$ and any node $u \notin V_s$, $I(\cdot)$ is submodular by verifying the following function:

$$I(V_{s'} \cup \{u\}) - I(V_{s'}) \geq I(V_s \cup \{u\}) - I(V_s)$$

If $c(\{u\}) \cap c(V_s) = \emptyset$, we have $I(V_s \cup \{u\}) = I(V_s) + I(u)$ and $I(V_{s'} \cup \{u\}) = I(V_{s'}) + I(u)$, which satisfies the equation. **Case (ii)**. Otherwise, we assume that $I(V_{s'} \cup \{u\})$ = $I(V_{s'}) + \triangle I(u|V_{s'})$ and $I(V_s \cup \{u\})$ = $I(V_s) + \triangle I(u|V_s)$, where $\triangle I()$ indicates the marginal gain by adding the new element u. Due to the non-negative nature of $c(V_s \backslash V_{s'})$, we have $|c(u) \cap c(V_s \backslash V_{s'})| \geq 0$, which indicates that $\triangle I(u|V_{s'}) \geq \triangle I(u|V_s)$, thus satisfying the inequality.

Following a similar analysis, we can show that $D(\cdot)$ is also monotone submodular. First, it is clear that the diversity function $D(\cdot)$ is monotone and non-negative due to the nature of set union. Second, to prove that $D(V^l_s)$ is submodular, we need to show that the function satisfies the diminishing returns property. Considering the same condition $V_{s'} \subset V_s$ and any node $u \notin V_s$, $D(V_{s'} \cup \{u\}) - D(V_{s'}) \geq D(V_s \cup \{u\}) - D(V_s)$. According to union property [38], since $V_{s'} \subset V_s$, the contribution of $r(u)$ to the union $\bigcup_{v \in V_{s'} \cup \{u\}} r(v)$ will be greater than or equal to its contribution to $\bigcup_{v \in V_s \cup \{u\}} r(v)$ due to overlapping elements in $r(u)$ that are already covered by V_s. This means that the increase in $\left|\bigcup_{v \in V_{s'} \cup \{u\}} r(v)\right|$ from adding u to $V_{s'}$ is at least as large as the increase in $\left|\bigcup_{v \in V_s \cup \{u\}} r(v)\right|$ from adding u to V_s, satisfying the diminishing returns property required for submodularity. Putting these together, the bi-criteria explainability measure $f(G^l_s)$ is a monotone submodular function.

The bi-criteria explainability measure $f(\cdot)$ is a submodular function maximizing relative influence $I(V^l_s)$ and embedding diversity $D(V^l_s)$. We then employ the approximation-preserving reduction to a Max-Sum Diversification problem [21] which indicates a greedy algorithm offering a $\frac{1}{2}$-approximation for this goal (without the post-processing phase in line 10-15).

Theorem 2. *Given a configuration $\mathcal{C}$ that specifies a single node v_t and a single source layer l, there is a $\frac{1}{2}$-approximation for the problem of explanation generation with model-slicing, which takes $O(k|G^l_{v_t}| \log |V^l_{v_t}| + L|V^l_{v_t}|^2)$ time.*

Proof: Given a configuration $\mathcal{C}$, with a specific test node v and a specific GNN layer l, we reformulate the problem as node selection problem, analogous to a counterpart as

a Max-Sum Diversification task [21], with a modified facility dispersion objective $f'(V_s) = \gamma \sum_{v \in V_s^l} |c(v)| + (1 - \gamma) \sum_{u,v \in V_s^l} \mathbf{1}[d(Z_u^l, Z_v^l) \geq \theta]$. Thus, given the approximation preserving reduction, we can map known results about Max-Sum Diversification to the diversification objective. First of all, we observe that maximizing the objective is **NP**-Hard, but there are known approximation algorithms for the problem [23]. Employing the greedy algorithm for Max-Sum Diversification yields a $\frac{1}{2}$-approximation. This achieves the desired $\frac{1}{2}$-approximation.

Time Cost. Slice(SS) takes $O(|G_{v_t}^l|)$ time to generate l-hop subgraph centered at v_t with breadth-first traversal (line 1). It requires $\lceil \frac{k}{2} \rceil$ iterations of explanatory nodes selection, where each iteration takes $O(T|V_{v_t}^l|)$ to identify the nodes with the maximum marginal gain (based on Equation 3). Here, T is the average time for calculating the marginal gain for each node $v \in V_{v_t}^l$. Next, it requires time cost in $O(|G_{v_t}^l| * log|V_{v_t}^l|)$ to induce connect sets and then to induce a connected subgraph to be further verified. Procedure Verify checks whether the subgraph G_s^l is an explanation in $O(l|G_{v_t}^l|)$ time. The post processing takes at most $|V_{v_t}^l|$ rounds and a verification cost in $O(l|V_{v_t}^l|)$ time, given the feature number per node is often small. Combining these components, as T and $O(l|G_{v_t}^l|)$ are relatively small compared to the dominant terms, the overall time cost is in $O(k|G_{v_t}^l| \log |V_{v_t}^l| + l|V_{v_t}^l|^2)$. Here, $l|V_{v_t}^l|^2$ comes from the post-processing stage (line 10-15, Alg 1). Empirically, such worst-case scenarios occur in only about 3% of instances, so the actual cost is much lower in practice.

With the above analysis, Theorem 2 follows.

Parallelization. In SliceGX, the explanation computation phase for each node at a specific layer is fully independent, providing a solid foundation for parallelization that enables the explanation process to scale efficiently with the number of available computing resources. To evaluate the scalability and efficiency of parallelization, we conduct experiments on a billion-scale synthetic dataset, demonstrating that SliceGX achieves significant speedup and effectively utilizes multi-core environments for large-scale graph explanation tasks.

B. Multi-nodes, Single-Source Layers

In Slice(SS), the explanations are generated for each test node sequentially, which is inefficient. To address this limitation, Slice(MS) employs an incremental approach, simultaneously selecting explanatory nodes for all test nodes V_T during *Generation Phase* in Algorithm 1.

Algorithm. The algorithm, denoted as Slice(MS) and illustrated as Algorithm 2, takes a configuration $\mathcal{C}$ as input, and generates a set of explanations $\mathcal{G}_s^l$ for the outputs $\mathcal{M}^{l_t}(G, V_T)$ of all test nodes V_T, at a specific layer l.

Initialization (lines 1-6). Algorithm 2 starts with setting $G^l(V_T)$ as all potential explanatory nodes for V_T because of data locality of GNN inference. Next, it introduces entry $\mathcal{V}_s[i]$ to store the "current" explanatory node set of the i-th element in V_T, and a boolean value $v.B[i]$ to represent whether a potential explanatory node $v \in G^l(V_T)$ is likely (i.e., a "candidate") to join $\mathcal{V}_s[i]$. For i-th element in V_T, denoted as v_{t_i}, $\mathcal{V}_s[i]$ is initialized to an empty set, indicating that no explanatory nodes have been selected for v_{t_i}, and each node $v \in G^l(v_{t_i})$ is a "candidate" explanatory node for v_{t_i}, i.e., $v.B[i] = true$. In addition, it maintains a global candidate explanatory node set V_s' (line 6), which is used to keep track of the remaining potential explanatory nodes in $G^l(V_T)$ that have not yet been selected.

Incremental generation phase (lines 7-17). Slice(MS) follows a pairwise node selection based on a greedy strategy similar to Slice(SS). When a new node $v \in \{v_1^*, v_2^*\}$ is introduced for maximum marginal gain corresponding to at least one test node, it uses v to update explanatory node sets $\mathcal{V}_s[]$. Specifically, if $v.B[i]$ is true, it indicates that v is a "promising" explanatory node for the i-th test node. Thus, the explanatory sets of such test nodes influenced by v can be updated all at once. When an explanatory node set of a test node in $\mathcal{V}_s[]$ is updated, it generates a subgraph G_s with connectivity. Next, it invokes Procedure IncVerify(MS) to test if G_s is an explanation for multiple test nodes in

Algorithm 2 : Slice(MS) (multi nodes, single source layer)

Input: a configuration $\mathcal{C} = (G, \mathcal{M}, V_T, \{l\}, l_t, k)$;
Output: a set of explanations $\mathcal{G}_s^l$ for each $v_t \in V_T$ w.r.t. $\mathcal{M}^{l_t}(G, v_t)$ at GNN layer l.

1: $G^l(V_T) = \bigcup_{v_t \in V_T} G^l(v_t)$ // union l-hop of V_T
2: Set $\mathcal{V}_s[i] = \emptyset$ for $i \in [0, |V_T| - 1]$
3: **for** $i \in [0, |V_T| - 1]$ **do**
4: **for** $v \in G^l(v_t)$, where v_t is the i-th element in V_T, **do**
5: set $v.B[i] = true$ // a boolean indicator
6: set $V_s' = G^l(V_T)$
7: **for** $i \in [0, |V_T| - 1]$ **do**
8: **while** $|\mathcal{V}_s[i]| < k$ and $V_s' \neq \emptyset$ **do**
9: select $\{v_1^*, v_2^*\} \in V_s'$ using pair-wise node selection in Slice(SS) (line 4) based on i-th element in V_T
10: **for** $j \in [0, |V_T| - 1]$ **do**
11: **if** $v_1^*.B[j] =$ true **then**
12: update $\mathcal{V}_s[j] = \mathcal{V}_s[j] \cup \{v_1^*\}$
13: **if** $v_2^*.B[j] =$ true **then**
14: update $\mathcal{V}_s[j] = \mathcal{V}_s[j] \cup \{v_2^*\}$
15: IncVerify(MS) $(\mathcal{V}_s[i], C)$
16: $V_s' = V_s' \backslash \{v_1^*, v_2^*\}$
17: **return** $\mathcal{G}_s^l$.

Procedure 2 IncVerify(MS) (V_s, C)

1: set G_s as the subgraph derived from node set V_s while maintaining connectivity
2: **for** $i \in [0, |V_T| - 1]$ **do**
3: **if** $\forall v \in V_s, v.B[i] = true$ **then**
4: **if** Verify(SS) $(G_s, C) = true$ **then**
5: $\mathcal{V}_s[i] = V_s$
6: $V_T = \{v_t | v_t \in V_T$, no explanation for $v_t\}$

V_T, which means that the nodes in G_s can be validated to determine whether it supports to be factual or counterfactual subgraphs for multiple test nodes in V_T, which is different from Algorithm 1. However, as each newly introduced explanatory node updates the explanatory sets $\mathcal{V}_s[]$ for multiple test nodes, this efficiency improvement comes at the cost of no longer strictly satisfying $\frac{1}{2}$-approximation guarantee, as it may introduce subtle deviations in the explanations of certain test nodes compared to those derived from Algorithm 1.

Procedure IncVerify(MS). The procedure verifies if the subgraph G_s induced by explanatory node set V_s serves as an explanation for multiple test nodes in V_T and efficiently updates the explanations for multiple test nodes. Specifically, for all node $v \in V_s$, if $v.B[i]$ is true, which means that V_s can serve as a candidate explanation for the i-th test node v_{t_i}, then it invokes Verify(SS) to verify if G_s is an explanation for test nodes that satisfy the verification condition in Section IV-A. If so, it updates $\mathcal{V}_s[i]$ as the nodes in G_s and reduces V_T to retain only those that have no explanation to further explore explanations for them in next loop.

Time cost. Slice(MS) employs a similar time cost analysis as Slice(SS), requiring $O(|G^l(V_T)|)$ time to generate the union l-hop subgraph centered on all targeted nodes V_T. Each node can be selected and updated only once for all V_T, necessitating $\lceil \frac{k}{2} \rceil$ iterations for selecting explanatory nodes. The time cost for inducing connected sets and creating a connected subgraph to be verified by IncVerify(MS) is $O(|G^l(V_T)| \log |V_T|)$. Post-processing follows the same procedure as Slice(SS). The key distinction is that Slice(MS) updates auxiliary structures for all test nodes V_T simultaneously during the explanatory node selection phase, thus reducing computation, with each node $v \in V_s$ used only once. The time complexity for this phase is $O(l|N^l(V_T)|^2)$, where $N^l(V_T)$ represents the set of neighbors for the test nodes within the l-hop neighborhood. Consequently, the overall time cost is $O(k|G^l(V_T)| \log |N^l(V_T)| + l|N^l(V_T)|^2)$.

C. Multi-nodes, Multi-Source Layers

We describe the Slice(MM) procedure, illustrated in Algorithm 3. Slice(MM) processes the layers $\mathcal{L}$ sequentially, starting from the last layer and moving downward (i.e, layer 1). For each layer, it performs two main steps: (1) It executes a "hop jumping" process that prunes the explanatory node set by removing nodes beyond l hops when at layer l. This layer-by-layer updating and pruning enhance efficiency by narrowing the search space to relevant nodes, thus avoiding unnecessary computations on irrelevant nodes for the current layer's output. (2) It updates a map $\mathcal{V}_B$ using Slice(MS), where each entry $\mathcal{V}_B[i][j]$ contains the best explanatory node set $V_s^{l_j}$ for the output $\mathcal{M}^{l_t}(G, v_{t_i})$ concerning $v_{t_i} \in V_T$ at layer l_j.

D. Making GNN Explanations "Queryable"

In this section, we provide SliceGXQ, a class of declarative, GNN exploratory queries to express the generation and access needs of SliceGX. The queries wrap our SliceGX algorithms,

Algorithm 3 : Slice(MM) (multi nodes, multi source layers)

Input: a configuration $\mathcal{C} = (G, \mathcal{M}, V_T, \text{Ł}, l_t, k)$;
Output: a set of explanations $\mathcal{G}_s = \{\mathcal{G}_s^1, ..., \mathcal{G}_s^L\}$ for each $v_t \in V_T$ and layer $l \in \text{Ł}$ w.r.t. $\mathcal{M}^{l_t}(G, v)$ at GNN layer l.

1: $G^L(V_T) = \bigcup_{v_t \in V_T} G^L(v_t)$
2: $\forall v_{t_i} \in V_T$, $\forall l_j \in \text{Ł}$, set $\mathcal{V}_B[i][j] = \emptyset$
3: sort Ł in a decreasing order
4: **for** $l_j \in$ Ł **do**
5: invoke Slice(MS) to update $V_B[][j]$ w.r.t. layer l_j
6: set $\mathcal{G}_s^{l_j}$ as a set of subgraphs derived from $V_B[][j]$
7: **for** $v_{t_i} \in V_T$ **do**
8: **if** v_{t_j} has no explanation at layer l_j **and** $l_j \neq$ Ł **then**
9: $V_B[i][j+1] = V_B[i] \backslash \{v | v \notin N^{l_{j+1}}(v_{t_i})\}$
10: **return** $\mathcal{G}_s$.

TABLE I
SliceGXQ QUERIES AND USER-FRIENDLY OPERATORS

SliceGXQ	Application
EXPLAIN ($\mathcal{M}$, $\{v\}$) At 3 FROM $\{G, \{1\}\}$ WHERE 4	GNN explanation with size bound k
EXPLAIN ($\mathcal{M}$, $\{v\}$) At 3 FROM $\{G, \emptyset\}$ WITH *diagnose*	GNN diagnose
EXPLAIN ($\mathcal{M}$, $\{v_1, v_2\}$) At 2 FROM $\{G, \emptyset\}$ WITH *interpret*	progressive GNN interpretation
EXPLAIN ($\mathcal{M}$, $\{v\}$) At 3 FROM $\{G, \{1, 2\}\}$ WHERE { ?review1 :connectedTo ?review2 . ?review2 :connectedTo ?review3 . ?review3 :connectedTo ?review1 . ?review1 :label "fraud" . ?review2 :label "fraud" . ?review3 :label "fraud" . FILTER (?review1 != ?review2 && ?review2 != ?review3) }	find explanations containing specific structures, such as a triangle with all fraud nodes
EXPLAIN ($\mathcal{M}$, $\{v_1, v_2\}$) At 3 FROM $\{G, \{1, 2\}\}$ ORDER BY *node.id* ASC	sort explanations by node id in ascending order

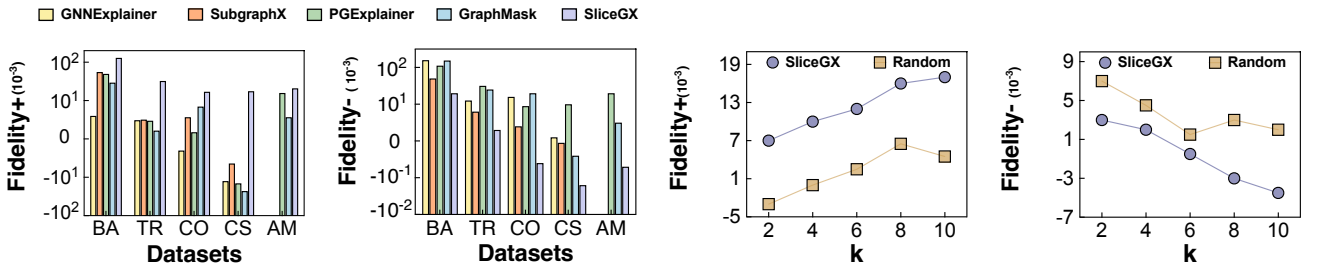
and can be easily expressed as a fragment of graph-native, structured SPARQL-like statements. A SliceGXQ query is in a following general form:

EXPLAIN ($\mathcal{M}$, V_T) **AT** l_t
FROM $\{G, \text{Ł}\}$
WHERE *P, k* **WITH** *mode*

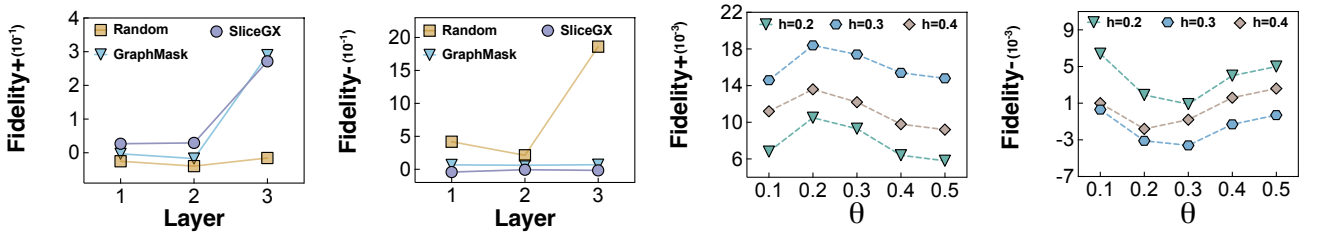
which specifies a configuration $\mathcal{C} = (G, \mathcal{M}, V_T, \text{Ł}, l_t, k)$. Optionally, it allows users to declare "shorthand" explanation refinement operation specified by MODE and P, where MODE declares explanation generation mode for specific scenario and P is a fragment of SPARQL pattern that refines the returned explanations via pattern matching. We showcase several SliceGXQ queries and corresponding applications (summarized in Table I), with the detailed processes provided in Appendix D.

VI. EXPERIMENTAL STUDY

We next evaluate SliceGX-based explanation algorithms. We aim to investigate the following questions. **RQ1**: Can SliceGX-based algorithms generate high-quality explanations for designated layers? **RQ2**: How fast can SliceGX generate explanations in a progressive manner? **RQ3**: What are the

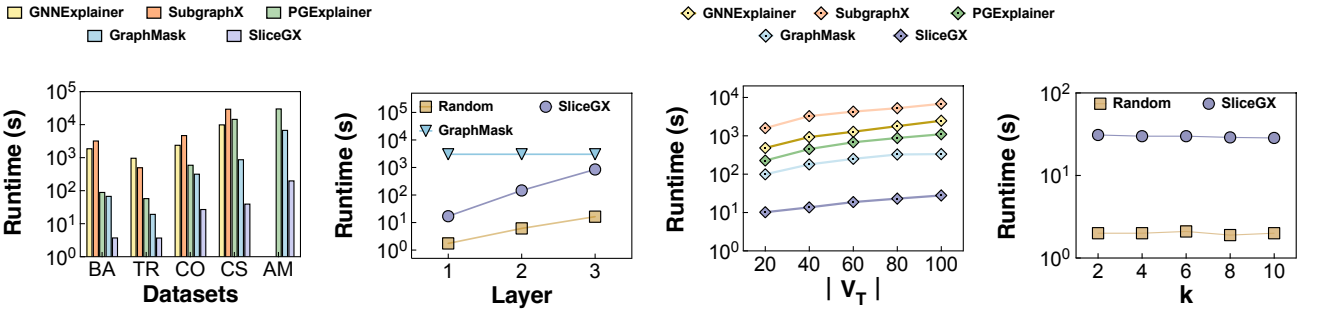


(a) Fidelity+: Overall performance (b) Fidelity-: Overall performance (c) Fidelity+: Varying size bound k (d) Fidelity-: Varying size bound k



(e) Fidelity+: Varying source layers (f) Fidelity-: Varying source layer (g) Fidelity+: Varying (h, θ) (h) Fidelity-: Varying (h, θ)

Fig. 5. Quality of explanations and impact of factors



(a) Efficiency: Overall (b) Efficiency: Varying source layer (c) Efficiency: Varying $|V_T|$ (d) Efficiency: Varying size bound k

Fig. 6. Efficiency of explanation generation and impact of factors

TABLE II
STATISTICS AND PROPERTIES OF DATASETS

Category	Dataset	# nodes	# edges	# node features	# classes
Synthetic	BA-shapes	700	4110	10	4
	Tree-Cycles	871	1950	10	2
Real-world	Cora	2708	10556	1433	7
	Coauthor CS	18333	163788	6805	15
	YelpChi	45954	7693958	32	2
	Amazon	1569960	264339468	200	107

impact of important factors to the quality and efficiency of explanation generation? We also illustrate the application of SliceGX-based explanation for model diagnosis. Our code and datasets are at: [1].

A. Experimental Setup

Datasets. We adopt six benchmark graph datasets, as summarized in Table II: **BA-shapes (BA) [18]**, **Tree-Cycles (TR) [18]**, **Cora (CO) [32]**, **YelpChi (YC) [15]**, **Coauthor CS (CS) [47]**, and **Amazon (AM) [57]**. These datasets cover synthetic and real-world graphs for tasks such as node classification, spam detection, and co-authorship analysis. We provide the details in Appendix B.

GNN classifiers. For all datasets, we have trained 3-layer Graph Convolutional Networks (GCNs) [32] for comparison. We used the Adam optimizer [14] with a learning rate of 0.001 for 2000 epochs. We also showcase that SliceGX can be applied to other types of GNNs. We report more details in Appendix C.1.

GNN explainers. We have implemented the following explainers: **(1)** our SliceGX algorithm. **(2)** Random selects and induces explanations by uniformly at random selection up to k arbitrary nodes as explanatory nodes from G. **(3)** GNNExplainer [54] uses mutual information to identify key edges and node features for instance-level classification. **(4)** SubgraphX [56] applies Monte Carlo tree search to find important subgraphs via node pruning for graph classification. **(5)** PGExplainer [36] models edges as conditionally independent Bernoulli variables, optimized for mutual information. **(6)** GraphMask [45] trains edge masks to prune irrelevant edges and generate layer-specific subgraphs. As GNN-LRP[46], Excitation-BP[42], and KerGNNs [17] are not model-agnostic and require access to the model's parameters during the explanation process, we exclude them from our comparison. We also provide the experimental results for Slice(MS) and Slice(MM) in Appendix C.2 to demonstrate

their overall performance and efficiency.

Evaluation metrics. We extend Fidelity, an established metric [30, 43, 55] for layer-wise explanations. Fidelity evaluates the faithfulness of explanations at layer l with the normalized difference between the output of a GNN over G and $G \setminus G_s^l$, where G_s^l is the explanation at layer l. (1) We define **Fidelity+** as:

$$Fidelity+ = \frac{1}{|V_T|} \sum_{v_t \in V_T} (\phi^l(G, v_t)_c - \phi^l(G \setminus G_s^l, v_t)_c) \quad (6)$$

where $\phi^l(\cdot)_c$ is the predicted likelihood of label c for v_t from $\mathcal{M}^l$, and c is the label assigned by $\mathcal{M}^l(G, v_t)$. Fidelity+ quantifies the impact of removing the explanation on label assignment, with higher values indicating better explanations. (2) Similarly, we define **Fidelity-** of G_s^l as:

$$Fidelity- = \frac{1}{|V_T|} \sum_{v_t \in V_T} (\phi^l(G, v_t)_c - \phi^l(G_s^l, v_t)_c) \quad (7)$$

Fidelity- at layer l quantifies how "consistent" the result of GNN is over G_s^l compared with its result at layer l over G. The smaller the value is, the "better".

GNNExplainer, SubgraphX, and PGExplainer do not support layer-wise explanation. We hence directly use their generated explanation subgraphs within the l-hop neighborhood of the test node as the explanation at layer l, to evaluate if they can generate explanations for the outputs at intermediate layers.

We also report the total time cost of explanation generation. For learning-based explainer, the cost includes the learning overhead.

B. Experimental Results

Exp-1: Quality of explanations. We evaluate explanation quality for target layer output, setting source layer Ł $= \{1\}$, target layer $l_t = 1$, and training 3-layer GCNs with $k = 10$ for all datasets. We sample 100 labeled nodes ($|V_T| = 100$) with known ground truth.

Overall Fidelity. Figs. 5(a)-5(b) show Fidelity+ and Fidelity- across all datasets, with ranges starting from the smallest value for relative comparison. SliceGX consistently improves Fidelity over other explainers (GNNExplainer, SubgraphX, PGExplainer, GraphMask) for progressive interpretation, particularly for non-final target layers. For instance, on BA with target layer 1, SliceGX outperforms others, demonstrating its ability to provide faithful intermediate-layer explanations, unlike methods focused solely on final outputs.

Impact of k. Varying k from 2 to 10, Figs. 5(c) and 5(d) show that both SliceGX and Random (omitting other explainers as they lack explicit k configuration) improve Fidelity with larger k, as more nodes and edges can be explored. SliceGX consistently outperforms Random in Fidelity+ and Fidelity-, validating its node selection strategy and alignment with Fidelity measures.

Impact of source layers. We vary source layers from 1 to 3 (Figs. 5(e), 5(f)) to assess robustness with fixed target layer 3. For Fidelity+, SliceGX and GraphMask outperform Random, with sensitivity increasing as source layers approach the output layer. This confirms: (i) GraphMask's effectiveness for final output interpretation, (ii) SliceGX's comparable or better performance without learning overhead, and (iii) Random's limited improvement due to random node selection. For Fidelity-, SliceGX and GraphMask are robust across layers, while Random is unstable due to its lack of guarantees on the objective. SliceGX consistently outperforms both, demonstrating its ability to generate high-quality layer-wise explanations.

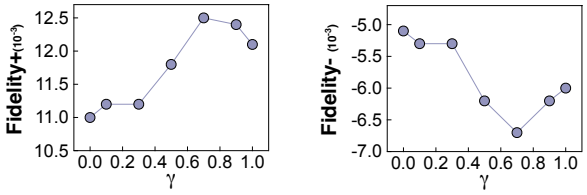
Impact of thresholds. We evaluate thresholds h (influence) and θ (embedding diversity). Figs. 5(g) and 5(h) show: (1) Proper embedding diversity (θ between 0.1 and 0.2) improves explanations by mitigating bias from similar nodes, but excessive diversity ($\theta > 0.2$) introduces noise and reduces quality. (2) Interestingly yet consistently, higher influence (h between 0.2 and 0.3) improves explanations, but overly influential nodes can introduce bias if their embeddings differ significantly from target nodes. SliceGX's bi-criteria objective balances influence and diversity, enabling high-quality layer-wise explanations and progressive inspection of GNN outputs. These results validate SliceGX's ability to explicitly generate and analyze layer-wise explanations, providing insights into the impact of influence and diversity on GNN outputs. The above results also justify our bi-criteria objective to allow SliceGX to strike a balance between influence and diversity over individual layers.

Impact of γ. We also evaluate the effect of the balance factor γ between relative influence and embedding diversity in Equation 3. As shown in Figs. 7(a) and 7(b), when γ is too small (i.e., $\gamma < 0.3$), the objective favors embedding diversity, potentially introducing noisy or weakly relevant nodes and degrading explanation quality. Conversely, when γ is too large (i.e., $\gamma > 0.8$), the objective overly prioritizes influence, which may lead to biased explanations dominated by highly influential but redundant nodes. SliceGX achieves the best performance when γ lies in the moderate range (i.e., $\gamma \in [0.6, 0.8]$), effectively balancing influence and diversity to yield high-quality explanations. These results further validate the design of our bi-criteria objective and demonstrate that SliceGX can flexibly adapt to different explanation needs by tuning the factor γ.

Exp-2: Efficiency of explanation generation. We next report the efficiency of SliceGX, compared with all other GNN explainers.

Overall performance. Using the same setting as in Figs. 5(a)-5(b), we report the time costs in Fig. 6(a). SliceGX consistently outperforms other GNN explainers, being 276, 482, 116, and 18 times faster than GNNExplainer, SubgraphX, PGExplainer, and GraphMask, respectively. GNNExplainer and SubgraphX face high overhead due to mask learning and tree search, respectively, and fail to complete on AM (after 24 hours). GraphMask performs better among baselines. We show that even with 3 layers, a common setting for GNNs, our method is still more efficient.

Impact of layers. As shown in Fig. 6(b), both SliceGX and Random take more time as the selected source layer ap-



(a) Fidelity+: varying γ (b) Fidelity-: varying γ

Fig. 7. Impact of γ on quality of explanations

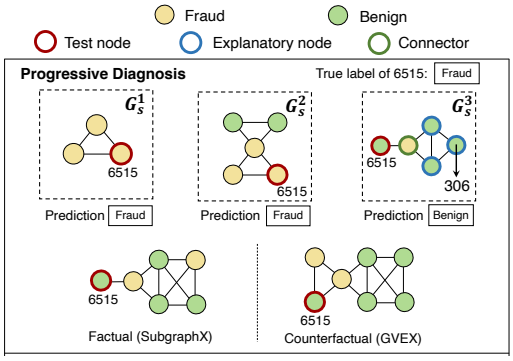


Fig. 8. Case study on progressive diagnose mode.

proaches the output one. This is consistent with that both exploits data locality and only explore l-hop neighbors for explanation generation. As l becomes larger, more nodes need to be explored. GraphMask is not sensitive to the selection of l because it learns edge masks using the original graph, and generates edge masks for all layers simultaneously.

Impact of V_T and k. As shown in Figs. 6(c) and 6(d), SliceGX maintains high efficiency and scales well with query loads, as confirmed by our experiments with varying the number of test nodes and parameter k.

Parallelization analysis. To evaluate the parallel efficiency and the scalability of SliceGX on large-scale graphs, we follow the graph generation procedure proposed by [54] to extend BA dataset to a billion-scale synthetic dataset BA_large with approximately 1 billion nodes and 1.34 billion edges. We randomly sample 300 test nodes and generate explanations for each layer of a 3-layer GCN model, resulting in a total of 900 independent explanation tasks. Since these tasks are mutually independent and stateless, they can be naturally distributed across multiple processes. Compared to the single-process setup, which takes approximately 12 hours to complete all explanation tasks, the 2-process version reduces the total runtime by approximately 1.4×, while the 4-process version achieves a nearly 2× speedup. These results confirm that SliceGX can effectively leverage multi-core environments, demonstrating high parallel efficiency for extremely large graph datasets.

Exp-3: Case study. We next showcase the application of SliceGX for diagnose mode. As shown in Fig. 8, assume a user aims to find "why" a GCN $\mathcal{M}$ misclassified node 6515 (with true label "Fraud") as prediction "Benign" and "when" (at which layer) it fails (Fig. 8). Using SliceGXQ in "progressive diagnosis" mode, explanations G_s^1, G_s^2, and G_s^3 reveal that $\mathcal{M}$ remains correct up to the second layer. Comparing G_s^2 and G_s^3, the error likely stems from the node 306 ("Benign") at hop 3, leading to a failure at layer 3 and biasing the inference. This helps fine-tune layer 3 (e.g., adjust weights) to improve accuracy. SliceGX can provide finer-grained, node- and layer-level explanations, whereas SubgraphX [56] and GVEX [11] – two state-of-the-art GNN explainability methods – only generate one-time, final output's explanation, which prevents them from tracing how errors propagate through intermediate layers pinpointing where the model fails, making it difficult to diagnose misclassifications.

TABLE III
TEST ACCURACY COMPARISON BEFORE AND AFTER MODEL DIAGNOSING. "BEFORE" IS THE ORIGINAL 3-LAYER GNN MODEL. "FINE-TUNING-1" AND "FINE-TUNING-2" REPRESENTS A FINE-TUNED MODEL WITH A SPECIFIC LAYER DEBUGGED AND REMOVED, RESPECTIVELY.

Dataset	Accuracy Before	Accuracy Fine-tuning-1	Accuracy Fine-tuning-2
BA-shapes	0.9586	**0.9743**	**0.9743**
Tree-Cycles	0.9288	**0.9323**	0.9219
Cora	0.9557	**0.9568**	0.9564
Coauthor CS	0.9669	0.9685	**0.9690**
YelpChi	0.8116	0.8189	**0.8453**
Amazon	0.6204	0.6278	**0.6511**

Exp-4: Case study2. In this case study, we show our explanations can facilitate model diagnosing and improvements. "Fine-tuning-1" first identifies a specific GNN layer using SliceGX as the primary source of classification errors. It then freezes all other layers and adjusts only the selected layer's weights while preserving the original architecture. As shown in Table III, this targeted approach consistently improves test accuracy across datasets. Alternatively, architecture optimization ("Fine-tuning-2") simplifies the model by completely removing the most problematic layer, thereby reducing the overall number of layers. After pruning, the remaining model is further fine-tuned by adjusting its parameters using the original parameters to adapt to the new architecture. "Fine-tuning-2" also demonstrates the effectiveness of SliceGX in model debugging and enhancing model performance.

VII. CONCLUSION

We introduced SliceGX, a layer-wise, novel GNN explanation framework that leverages model slicing to generate explanations from selected source layers for output at target layers. We formalized explanation generation as a novel bi-criteria optimization problem, and introduced novel and efficient algorithms for both single-node, single-source and multi-node, multi-source settings. Our theoretical and experimental results demonstrate the effectiveness, efficiency, and scalability of SliceGX algorithms, as well as their practical utility for model debugging in real-world applications. A future topic is to extend SliceGXQ to more expressive language and enrich SliceGX to support more explanatory operators.

References

[1] 2025. Code. https://github.com/TingtingZhu-ZJU/SliceGX.

[2] Julius Adebayo, Michael Muelly, Ilaria Liccardi, and Been Kim. 2020. Debugging tests for model explanations. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[3] Waqas Ali, Muhammad Saleem, Bin Yao, Aidan Hogan, and Axel-Cyrille Ngonga Ngomo. 2021. A survey of RDF stores & SPARQL engines for querying knowledge graphs. *The VLDB Journal* 31, 3 (2021), 1–26.

[4] Marcelo Arenas. 2024. A data management approach to explainable AI. In *Companion of the 43rd Symposium on Principles of Database Systems*. 1–3.

[5] Marcelo Arenas. 2024. A Data Management Approach to Explainable AI. In *Symposium on Principles of Database Systems*. 1–3.

[6] Mohit Bajaj, Lingyang Chu, Zi Yu Xue, Jian Pei, Lanjun Wang, Peter Cho-Ho Lam, and Yong Zhang. 2021. Robust counterfactual explanations on graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 5644–5655.

[7] Arnaud Blouin, Benoît Combemale, Benoit Baudry, and Olivier Beaudoux. 2011. Modeling model slicers. In *International Conference on Model Driven Engineering Languages and Systems*. 62–76.

[8] Shaofeng Cai, Gang Chen, Beng Chin Ooi, and Jinyang Gao. 2019. Model slicing for supporting complex analytics with elastic inference cost and resource constraints. *Proc. VLDB Endow.* 13, 2 (2019), 86–99.

[9] Oana-Maria Camburu, Eleonora Giunchiglia, Jakob Foerster, Thomas Lukasiewicz, and Phil Blunsom. 2019. Can I trust the explainer? Verifying post-hoc explanatory methods. In *NeurIPS Workshop on Safety and Robustness in Decision Making*.

[10] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable graph neural networks via bidirectional propagation. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[11] Tingyang Chen, Dazhuo Qiu, Yinghui Wu, Arijit Khan, Xiangyu Ke, and Yunjun Gao. 2024. View-based explanations for graph neural networks. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–27.

[12] Yu Christine Chen, Jianhui Wang, Alejandro D Domínguez-García, and Peter W Sauer. 2015. Measurement-based estimation of the power flow Jacobian matrix. *IEEE Transactions on Smart Grid* 7, 5 (2015), 2507–2515.

[13] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2018. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *SPM* 35, 1 (2018), 126–136.

[14] P Kingma Diederik. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

[15] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S. Yu. 2020. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *ACM International Conference on Information and Knowledge Management (CIKM)*. 315–324.

[16] Alexandre Duval and Fragkiskos D Malliaros. 2021. Graphsvx: Shapley value explanations for graph neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*. 302–318.

[17] Aosong Feng, Chenyu You, Shiqiang Wang, and Leandros Tassiulas. 2022. Kergnns: Interpretable graph neural networks with graph kernels. In *Proceedings of the AAAI conference on artificial intelligence*. 6614–6622.

[18] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

[19] 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* 35, 8 (2022), 8687–8698.

[20] Floris Geerts. 2023. A query language perspective on graph learning. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*. 373–379.

[21] Sreenivas Gollapudi and Aneesh Sharma. 2009. An axiomatic approach for result diversification. In *International conference on World Wide Web (WWW)*. 381–390.

[22] Shurui Gui, Hao Yuan, Jie Wang, Qicheng Lao, Kang Li, and Shuiwang Ji. 2023. Flowx: Towards explainable graph neural networks via message flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 7 (2023), 4567–4578.

[23] Refael Hassin, Shlomi Rubinstein, and Arie Tamir. 1997. Approximation algorithms for maximum dispersion. *Operations research letters* 21, 3 (1997), 133–137.

[24] Ke-Jou Hsu, Ketan Bhardwaj, and Ada Gavrilovska. 2019. Couper: DNN model slicing for visual analytics containers at the edge. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing (SEC)*. 179–194.

[25] 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* 35, 7 (2022), 6968–6972.

[26] Alexey Ignatiev. 2020. Towards trustable explainable AI. In *International Joint Conference on Artificial Intelligence*. 5154–5158.

[27] Md Tauhid Bin Iqbal, Abdul Muqeet, and Sung-Ho Bae. 2022. Visual interpretation of CNN prediction through layerwise sequential selection of discernible neurons. *IEEE Access* 10 (2022), 81988–82002.

[28] Minseok Jeon, Jihyeok Park, and Hakjoo Oh. 2024.

PL4XGL: A Programming Language Approach to Explainable Graph Learning. *Proceedings of the ACM on Programming Languages* 8, PLDI (2024), 2148–2173.

[29] Wei Ju, Siyu Yi, Yifan Wang, Zhiping Xiao, Zhengyang Mao, Hourun Li, Yiyang Gu, Yifang Qin, Nan Yin, Senzhang Wang, et al. 2024. A Survey of Graph Neural Networks in Real world: Imbalance, Noise, Privacy and OOD Challenges. *CoRR* (2024).

[30] Jaykumar Kakkad, Jaspal Jannu, Kartik Sharma, Charu Aggarwal, and Sourav Medya. 2023. A survey on explainability of graph neural networks. *IEEE Data Eng. Bull.* 46, 2 (2023), 35–63.

[31] Samir Khuller, Anna Moss, and Joseph Seffi Naor. 1999. The budgeted maximum coverage problem. *Information processing letters* 70, 1 (1999), 39–45.

[32] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

[33] Anna Langedijk, Hosein Mohebbi, Gabriele Sarti, Willem Zuidema, and Jaap Jumelet. 2024. DecoderLens: Layerwise Interpretation of Encoder-Decoder Transformers. In *Findings of the Association for Computational Linguistics: NAACL 2024*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.).

[34] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2019), 1475–1488.

[35] Ana Lucic, Maartje A Ter Hoeve, Gabriele Tolomei, Maarten De Rijke, and Fabrizio Silvestri. 2022. Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 4499–4511.

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

[37] Ge Lv and Lei Chen. 2023. On data-aware global explainability of graph neural networks. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3447–3460.

[38] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming* 14 (1978), 265–294.

[39] Tamara Pereira, Erik Nascimento, Lucas E Resck, Diego Mesquita, and Amauri Souza. 2023. Distill n' Explain: explaining graph neural networks using simple surrogates. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 6199–6214.

[40] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)* 34, 3 (2009), 1–45.

[41] Guilherme Perin, Sengim Karayalcin, Lichao Wu, and Stjepan Picek. 2022. I know what your layers did: Layer-wise explainability of deep learning side-channel analysis. *Cryptology ePrint Archive* (2022).

[42] 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.

[43] Mario Alfonso Prado-Romero, Bardh Prenkaj, Giovanni Stilo, and Fosca Giannotti. 2024. A survey on graph counterfactual explanations: definitions, methods, evaluation, and research challenges. *Comput. Surveys* 56, 7 (2024), 1–37.

[44] Tilman Räuker, Anson Ho, Stephen Casper, and Dylan Hadfield-Menell. 2023. Toward transparent AI: A survey on interpreting the inner structures of deep neural networks. In *Secure and Trustworthy Machine Learning (SaTML)*. 464–483.

[45] 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 (ICLR)*.

[46] Thomas Schnake, Oliver Eberle, Jonas Lederer, Shinichi Nakajima, Kristof T Schütt, Klaus-Robert Müller, and Grégoire Montavon. 2021. Higher-order explanations of graph neural networks via relevant walks. *IEEE transactions on pattern analysis and machine intelligence* 44, 11 (2021), 7581–7596.

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

[48] Ihsan Ullah, Andre Rios, Vaibhav Gala, and Susan Mckeever. 2021. Explaining deep learning models for tabular data using layer-wise relevance propagation. *Applied Sciences* 12, 1 (2021), 136.

[49] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations (ICLR)*.

[50] Baowen Xu, Ju Qian, Xiaofang Zhang, Zhongqiang Wu, and Lin Chen. 2005. A brief survey of program slicing. *ACM SIGSOFT Software Engineering Notes* 30, 2 (2005), 1–36.

[51] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *International Conference on Learning Representations (ICLR)*.

[52] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*. PMLR, 5453–5462.

[53] Han Yang, Junyuan Fang, Jiajing Wu, Dan Li, Yaonan

Wang, and Zibin Zheng. 2025. Soft label enhanced graph neural network under heterophily. *Knowledge-Based Systems* 309 (2025), 112861.

[54] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. GnnExplainer: Generating explanations for graph neural networks. In *Advances in neural information processing systems (NeurIPS)*. 9240–9251.

[55] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. 2022. Explainability in graph neural networks: A taxonomic survey. *IEEE transactions on pattern analysis and machine intelligence* 45, 5 (2022), 5782–5799.

[56] 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*. 12241–12252.

[57] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations (ICLR)*.

[58] Lingze Zeng, Naili Xing, Shaofeng Cai, Gang Chen, Beng Chin Ooi, Jian Pei, and Yuncheng Wu. 2025. Powering in-database dynamic model slicing for structured data analytics. *Proc. VLDB Endow.* (2025).

[59] Wentao Zhang, Zhi Yang, Yexin Wang, Yu Shen, Yang Li, Liang Wang, and Bin Cui. 2021. Grain: Improving data efficiency of graph neural networks via diversified influence maximization. *Proc. VLDB Endow.* 14, 11 (2021), 2473–2482.

[60] Xingyi Zhang, Jinchao Huang, Fangyuan Zhang, and Sibo Wang. 2024. FICOM: an effective and scalable active learning framework for GNNs on semi-supervised node classification. *The VLDB Journal* 33, 5 (2024), 1–20.

[61] 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.

[62] Ziqi Zhang, Yuanchun Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. 2020. Dynamic slicing for deep neural networks. In *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 838–850.

[63] Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. 2020. Database meets artificial intelligence: A survey. *IEEE Transactions on Knowledge and Data Engineering* 34, 3 (2020), 1096–1116.

APPENDIX A: DISCUSSION

A.1. In this work, we propose a sliced model, which is derived from a pre-trained GNN model. As described in Section IV-A, an l-sliced model $\mathcal{M}^l$ consists of two primary components: the feature extractor (denoted as f_1^l) and the predictor (denoted as f_2). In practice, for an l-sliced model of a pre-trained GNN $\mathcal{M}$ with L layers, the f_1^l component is constructed by taking the first l pre-trained GNN layers to capture hidden features from the graph data. Following the f_1^l component, it connects the original MLP (Multi-Layer Perceptron) layers of the GNN, constituting the f_2 part of our l-sliced model.

There are three options to construct an l-sliced model:

Option 1: Retrain both f_1^l and f_2;

Option 2: Freeze pre-trained f_1^l and retrain f_2; and

Option 3: Directly use pre-trained f^l and f_2 (adopted by SliceGX).

Compared to the first two options, we favor option 3 in this paper for three reasons: (a) The original model $\mathcal{M}$ can be treated as a black box as we focus solely on integrating layers without knowledge of internal workings of $\mathcal{M}$; (b) it saves time when directly utilizing the pre-trained components instead of engaging in time-consuming retraining processes; and (c) even if choosing option 1 or option 2 to retrain the model, inconsistencies in the final outputs could still exist.

As option 3 also generates a "new" model $\mathcal{M}^l$ with potential inconsistencies, we introduce *guard condition* for explanation at layer l, including "factual" and "counterfactual," to mitigate this issue (Section IV-A). For instance, by ensuring that the prediction of a test node v_t using an explanation subgraph G_s^l in the sliced model $\mathcal{M}^l$ is consistent with the prediction of v_t using original graph G in both $\mathcal{M}$ and $\mathcal{M}^l$, the explanation generated by option 3 remains "faithful" to the original model, even without any retraining. This is the rationale behind the necessity of the guard condition. However, as both Option 1 and Option 2 are more likely to violate the guard condition, it becomes difficult to generate high-quality explanations that stays faithful to the original model's behavior. The main reasons are as follows: Although option 1 enables the sliced model $\mathcal{M}^l$ to fit the original data better, it poses a substantial risk of altering the original model's behavior. As our primary objective is to generate explanations based on the original GNN model, retraining both components would create a "new" model that may not capture the characteristics of the original architecture and potentially change the inference process. Similarly, option 2 might make the new sliced model $\mathcal{M}^l$ achieve closer accuracy to the final output of the original GNN, but it still introduces inconsistencies in model inference.

Based on derived sliced models, the design of SliceGX is inherently grounded in both the predictions of the original model and the sliced model in terms of guard condition, rather than treating the sliced model as an independent entity. However, for existing methods, such as GNNExplainer [54] and SubgraphX [56], even when applying directly to the sliced model, their explanations are not high-quality to provide truly progressive explanations that are faithful to both original model and sliced model, as they neglect the critical logical connection between the sliced model and the original model and are only accountable to the sliced model.

Example 6. *Consider a pre-trained GNN-based node classifier $\mathcal{M}$ with 3 layers, $\mathcal{M}^2$ is a 2-sliced model of $\mathcal{M}$. In Fig. 9 (upper part), G_s^l is an explanation generated by* SubgraphX *directly for sliced model $\mathcal{M}^2$, ensuring that the prediction of*

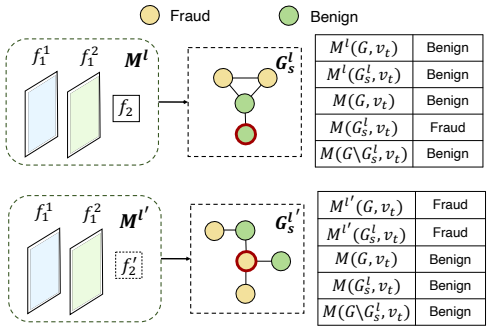
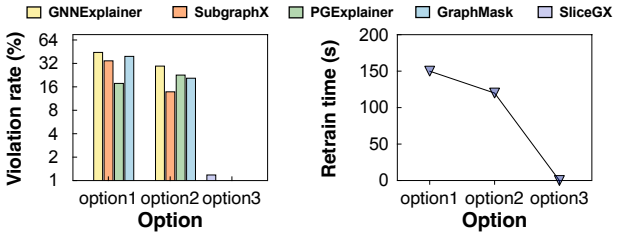


Fig. 9. An example of explanation generated by SubgraphX based on sliced model $\mathcal{M}^l$ and $\mathcal{M}^{l'}$ ($l = 2$), where $\mathcal{M}^l$ is obtained by option 3 (SliceGX) and $\mathcal{M}^{l'}$ is obtained by option 2. The table on the right presents the corresponding conditions for the factual and counterfactual cases of explanations generated in both the original model $\mathcal{M}$ and the sliced model $\mathcal{M}^l$.



(a) Violation rate (b) Retrain time

Fig. 10. Violation rate and retain time for three options

the test node v_t using explanation subgraph G_s^l is consistent with the prediction of v_t using original graph G in sliced model $\mathcal{M}^2$. However, such an explanation fails to be "faithful" to the original model $\mathcal{M}$ according to the discussion above, as it does not satisfy either the "factual" or "counterfactual" guard condition in Section IV-A. Specifically, the factual condition (a) is violated because the prediction of v_t in model $\mathcal{M}$ using graph G is not consistent with its prediction using the explanation subgraph G_s^l, i.e., $\mathcal{M}(G, v_t) \neq \mathcal{M}(G_s^l, v_t)$. Similarly, the counterfactual condition (a) is not satisfied as $\mathcal{M}(G, v_t) = \mathcal{M}(G \setminus G_s^l, v_t)$.

Similarly, $\mathcal{M}^{2'}$ is a 2-sliced model of $\mathcal{M}$ derived from option 2 and $G_s^{l'}$ is an explanation generated by SubgraphX *directly for the sliced model $\mathcal{M}^{2'}$ in Fig. 9 (lower part). Such an explanation also fails to be "faithful" to the original model $\mathcal{M}$ according to Section IV-A, as the prediction made by the original model is not aligned with that of the newly trained sliced model for the same graph G, i.e., $\mathcal{M}(G, v_t) \neq \mathcal{M}^{l'}(G, v_t)$, which violates factual condition (b). Also, the counterfactual condition (a) is not satisfied as $\mathcal{M}(G, v_t) = \mathcal{M}(G \setminus G_s^l, v_t)$. This inconsistency confirms the inherent issue with Option 2.*

The above example illustrates that when applying existing explainability methods to a sliced model with option 2 or option 3, they fail to satisfy the guard conditions because their objective functions are designed specifically for the output of the current model, and are difficult to adjust for guard conditions, making them less likely to remain faithful to both the original model $\mathcal{M}$ and the sliced model $\mathcal{M}^l$ simultaneously. Thus, compared to other explainers (Table IV), it demonstrates that SliceGX is a novel approach that can provides progressive explanations with model-slicing.

To further investigate inconsistency mentioned above, we compare the three options on CO dataset, evaluating both the frequency of guard condition violations and the retraining time required at layer 1. For option 1 and 2, we show results from existing explainers such as GNNExplainer and SubgraphX, while option 3 is specifically designed for SliceGX. As shown in Figs. 10(a) and 10(b), SliceGX yields the fewest guard condition violations with no retraining overhead. In contrast, explainers under Option 1 or 2 not only incur additional retraining time but also frequently generate explanations that significantly violate the guard condition—resulting in unfaithful explanations that do not accurately reflect the model's inference process.

A.2 Explanations for progressive interpretation and progressive diagnosis. Current GNN explanations are one-time and typically highlight the most important features for the final predictive output, while ignoring progressive explanation needs during inference. We extend the notion of explanation with model-slicing to two scenarios below, and elaborate their applications.

Progressive diagnosing. In diagnosing cases, GNNs may aggregate detrimental information at specific layers, potentially resulting in incorrect classification results during the inference process [29]. When a test node is classified with a wrong label, there is often a critical question: "*Which pieces of information in a specific layer impact the model's incorrect decision-making*"? To facilitate deeper analysis, we specify helpful explanations as those that are crucial for "wrong" results in a sliced model.

Formally, in the "diagnose mode", given a graph $G = (V, E)$, a GNN $\mathcal{M}$ with L layers, a test node $v_t \in V_T$, and a specific layer l ($1 \leq l \leq L$), G_s^l is a *diagnosis explanation* for $\mathcal{M}(G, v_t)$ if

- (a) $\mathcal{M}(G, v_t) \neq c^*$, where c^* is the ground truth label; *and*
- (b) $\mathcal{M}(G, v_t) = \mathcal{M}^l(G, v_t) = \mathcal{M}^l(G_s^l, v_t)$ (*"factual" at layer l*).

That is, the explanation G_s is useful for "diagnosing" the output as it ensures v_t to be *consistently* assigned the same "wrong" label by both $\mathcal{M}$ and $\mathcal{M}^l$ at layer l. Intuitively, if it exists, a diagnosis explanation G_s^l at layer l clarifies "why" the inference gets a "wrong" prediction and "when" (at which layer) it has made such a mistake.

Example 7. *Consider a pre-trained* GNN*-based node classifier $\mathcal{M}$ with 3 layers, the test node v_t with correct label "Fraud" is assigned the prediction label "Fraud", "Benign", and "Benign" at layer 1, layer 2, and layer 3, respectively, shown in Fig. 11. Here G_s^2 is a factual explanation of $\mathcal{M}(G, v_t)$ at layer 2 to reveal that (1) v_t is consistently assigned the same "wrong" label "Benign" by both $\mathcal{M}$ and $\mathcal{M}^2$.*

TABLE IV

COMPARISON OF SliceGX WITH STATE-OF-THE-ART GNN EXPLAINERS. "LEARNING-BASED" DENOTES WHETHER LEARNING OF NODE/EDGE MASK IS REQUIRED, "OUTPUT" SPECIFIES THE EXPLANATION STRUCTURE (N/NF/E: NODE/NODE FEATURES/EDGE), "MODEL-AGNOSTIC" INDICATES IF THE METHOD TREATS GNNs AS A BLACK-BOX (E.G., THE PARAMETERS OF GNNs ARE NOT REQUIRED), "SLICEABLE" MEANS IF THE GNN EXPLAINER IS DIRECTLY APPLICABLE FOR LAYER-SPECIFIC EXPLANATION, AND "USER-FRIENDLY QUERYING" MEANS IF THE METHOD SUPPORTS QUERY MECHANISM WITH HIGH USABILITY TO ALLOW USER TO DIRECTLY CONFIGURE LAYER-WISE EXPLANATION WITH INPUT AND TARGETED MODEL LAYERS

Explainability Methods	LEARNING-BASED	OUTPUT	MODEL-AGNOSTIC	SLICEABLE	USER-FRIENDLY QUERYING
SubgraphX [56]	✗	Connected Subgraph	✓	✗	✗
GNNExplainer [54]	✓	Disconnected E/NF set	✓	✗	✗
PGExplainer [36]	✓	Disconnected E set	✓	✗	✗
GraphMask [45]	✓	Disconnected E set	✓	✗	✗
GNN-LRP [46]	✗	Disconnected E set	✗	✓	✗
Excitation-BP [42]	✗	Disconnected N set	✗	✓	✗
KerGnns [17]	✓	Connected Subgraph	✗	✓	✗
SliceGX **(this work)**	✗	Connected Subgraph	✓	✓	✓

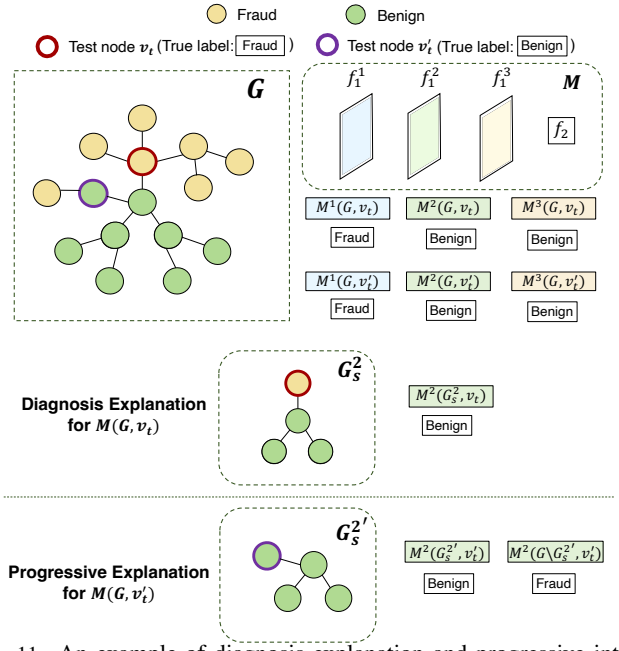


Fig. 11. An example of diagnosis explanation and progressive interpretation (G_s^2 is a diagnosis explanation for $\mathcal{M}(G, v_t)$ and $G_s^{2'}$ is a progressive interpretation for $\mathcal{M}(G, v_t{}')$)

Therefore, layer 2 can provide clues about why it is classified to the final incorrect label; (2) by keeping consistency with the prediction of v_t in G via $\mathcal{M}^2$, G_s^2 demonstrates its ability to be responsible for the intermediate layer output.

We also observe no diagnosis explanation at layer 1. Indeed, we can only identify subgraphs contributing to the intermediate output of "Fraud". However, these subgraphs are not helpful for diagnosis and are not responsible for the final erroneous result as the intermediate output at layer 1 is already correct.

Progressive interpretation. In interpretation case, "*What data one needs to inspect at a layer that may clarify a* GNN*'s final result?*" is essential for understanding the internal inference process of GNNs. In interpreting model output, simplicity is often preferred due to cognitive bandwidth of users and tolerance of time cost in explanation generation. Existing approaches are often expensive and may produce large, complex explanations. We characterize light-weighted explanations enabled by model slices, which incur much smaller cost and delay time. Moreover, this process can be interrupted at any moment for ad-hoc inspection.

Formally, in the "interpretation mode", given a graph $G = (V, E)$, a GNN $\mathcal{M}$ with L layers, a test node $v_t \in V$, and a specific layer l ($1 \leq l \leq L$), a subgraph G_s^l of G is a *progressive interpretation* for $\mathcal{M}$ if

- (a) $\mathcal{M}(G, v_t) = \mathcal{M}^l(G, v_t)$; and
- (b) *one* of the following conditions holds: (i) $\mathcal{M}(G, v_t) = \mathcal{M}^l(G_s^l, v_t)$ (*"factual" at layer l*), or (ii) $\mathcal{M}(G, v_t) \neq \mathcal{M}^l(G \setminus G_s^l, v_t)$ (*"counterfactual" at layer l*).

Here, $G \setminus G_s^l$ is the subgraph obtained by removing all the edges in G_s^l from G. Intuitively, an explanation G_s^l highlights the fraction of G and their associated features that are aggregated during the current inference process at the sliced model $\mathcal{M}^l$ that is responsible for the final result. Specifically, the explanation can be used to answer the question *"What data one needs to inspect at layer l that may clarify a* GNN *$\mathcal{M}$'s final result?"*. Indeed, G_s^l ensures that either (1) a consistent label will be assigned to v_t by the sliced model $\mathcal{M}^l$ at layer l ("factual"), or (2) if "removed" from G, $\mathcal{M}^l$ *would* assign the remaining fraction of G a different label ("counterfactual").

Example 8. *Continue with Example 7, the test node $v_t{}'$ (with true label "Benign") is assigned the predicted label "Fraud", "Benign", and "Benign" at layer 1, layer 2, and layer 3, respectively. $G_s^{2'}$ is a progressive interpretation of $\mathcal{M}(G, v_t')$ at layer 2 with both "factual" and "counterfactual" constraints: (1) Similar to Example 7, $G_s^{2'}$ is consistent with the intermediate output at layer 2, and v_t' is assigned the same label "Benign" by both $\mathcal{M}^2$ and $\mathcal{M}$; (2) as removing all edges in $G_s^{2'}$ from G while keeping all nodes, the prediction label of $v_t{}'$ by $\mathcal{M}^2$ is different from its final prediction.*

Remarks. We remark that the two modes are orthogonal: One can use interpretation even when $\mathcal{M}$ has a wrong label assigned to a test node v_t. In such cases, progressive interpretation serves as a case for "progressive diagnose".

APPENDIX B: DATASETS

We adopt six benchmark graph datasets [18], as summarized in Table II. (1) **BA-shapes** (BA) [18] is a synthetic

Barabási-Albert (BA) graphs [18]. The graph is generated by augmenting a set of nodes with structured subgraphs (motifs). A benchmark task for BA is node classification, where the nodes are associated with four numeric classes. Since the dataset does not originally provide node features, we follow PGExplainer [36] to process the dataset and derive the node features as 10-dimensional all-one vectors. (2) **Tree-Cycles (TR) [18]** are synthetic graphs obtained by augmenting 8-level balanced binary trees with six-node cycle motifs. The dataset includes two classes: one for the nodes in the base graph and another for the nodes within the defined motifs. Similar to BA-shapes, we process the dataset to derive the node features. (3) **Cora (CO) [32]** is a widely-used citation network. The nodes represent research papers, while the edges correspond to citation relationships. Each node is labeled with a topic for multi-label classification analysis. **(4) YelpChi(YC) [15] (CS)** is a binary classification dataset designed for spam reviews detection. It consists of product reviews categorized as "Benign User" or "Fraudster". Each node represents a review, while edges capture relationships between reviews, such as shared users, products, re- view text, or temporal connections. **(5) Coauthor CS [47] (CS)** is a co-authorship graph based on the Microsoft Academic Graph. The nodes include authors and papers, interlinked with co-authorship relations. The node features represent paper keywords, and each author is assigned a class label indicating his or her most active field of study. **(6) Amazon [57] (AM)** is a co-purchasing network, where each node is a product, and an edge between two products means they are co-purchased. The text reviews of the products are processed to generate 4-gram features, which are reduced to 200-dimensional vectors using SVD, serving as the node features. The labels represent product categories, e.g., books, movies.

Appendix C: Additional Experimental Results

C.1 Different types of GNNs. We showcase that SliceGX can be also applied to different types of GNNs, including GCN [32], GIN [51], and GAT [49]. In experimental setting, the hyperparameters of GIN, including the learning rate and training epochs, closely follow the setting of GCN. For GAT, we use 10 attention heads with 10 dimensions each, and thus 100 hidden dimensions.

To ensure a fair comparison across different architectures, we made minor adjustments to the baselines to enable them to adapt to these GNN variants. We report the results of main experiments to evaluate the overall quality of explanations generated by different explainers in CO dataset shown in Figs. 12(a) and 12(b). The other configurations align with the same settings in section VI-B. The results demonstrate the high quality of explanations generated by SliceGX across different models compared to other explainers such as GNNExplainer [54] and SubgraphX [56].

C.2 Performance of Slice(MS) and Slice(MM). We also evaluate the overall performance and efficiency over layers by two extended algorithms of SliceGX (Slice(SS)): Slice(MS) and Slice(MM). We set source layer $\text{Ł} = \{1, 2, 3\}$, target layer $l_t = 3$, trained GCN with layer number $L = 3$, and explanation size bound $k = 10$ by default for the CO dataset. We use the same 100 test nodes ($|V_T| = 100$).

Figs. 12(c) and 12(d) illustrate the overall performance including fidelity+ and fidelity-, and Fig. 12(e) presents the time for generating explanations over layers. The results indicate that compared to SliceGX, Slice(MS) achieves faster generation speed while almost maintaining original performance due to its global updating approach. Meanwhile, Slice(MM) utilizes a "hop jumping" process and eliminates some redundant greedy selection steps, which is the most time-efficient.

To evaluate the efficiency of the query processing, we report the progressive running time of SliceGX in the interpretation mode, i.e., the average time needed to update an explanation between two consecutive layers. A lower running time generally indicates lower computational complexity to process a query by the explainer, making it more suitable for real-time applications. Using the same GNN model and 100 test nodes for CO dataset as in Section VI, we track the average response time taken for generating explanations over layers by SliceGX. By setting explanation size bound $k = 10$, the average running time for each test node between consecutive layers is approximately 0.5 seconds for Slice(MS) and Slice(MM). The result indicates the ability of SliceGX to handle queries without significant computational overhead, showing its feasibility in generating explanations progressively. This is a desirable property for online explanation generation of GNN applications over data streams such as traffic analysis, social networks, and transaction networks.

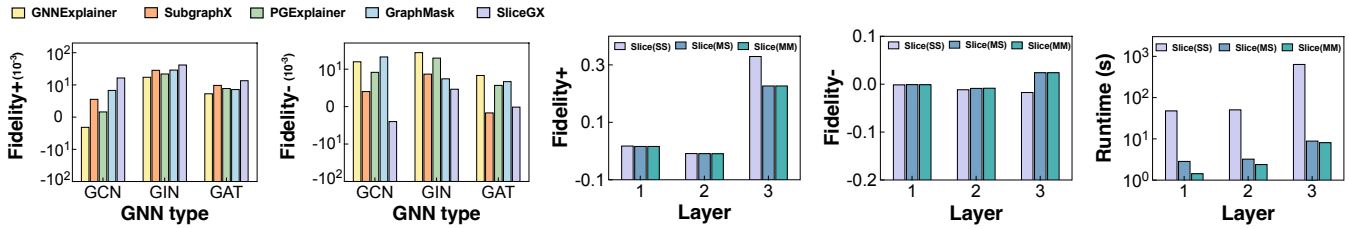
Appendix D: User-friendly Querying

In this section, we provide a detailed analysis of the queries in Table I, highlighting their evaluation processes and applications for better understanding of SliceGXQ 's functionality.

Q1: GNN explanation with size bound k . Consider a graph G, a GNN $\mathcal{M}$, a target layer 3, and a test node v, this query specifies a configuration $C = (G, \mathcal{M}, \{v\}, \{1\}, 3, 4)$, which aims to generate an optimal explanation with at most 4 explanatory nodes from each model slice $\mathcal{M}^l$ with selected layer $l \in \{1\}$ for the result $\mathcal{M}^3(G, v)$. The SliceGX algorithm is then executed using configuration C to derive the explanation result.

Q2: GNN explanation for diagnosing mode. This query is designed to generate progressive diagnosing explanations, aimed at facilitating a deeper analysis: *"At which layer the inference leads to a "wrong" prediction, and which pieces of information impact the model's incorrect decision-making for node v?"* The execution of the query is similar to Q1, with the key difference being the invocation of different verify operator functions, as provided in A.2.

Q3: GNN explanation for interpretation mode. This query aims to identify the key pieces of information to gain insights into the predictions of v_1 and v_2 at layer 2. Its execution follows a similar approach to Q1, with the main distinction



(a) Fidelity+ over GNNs (b) Fidelity- over GNNs (c) Fidelity+ over layer (d) Fidelity- over layer (e) Runtime($|V_T| = 100$)

Group1: Fidelity (different types of GNNs) **Group2: Performance of Slice(MS) and Slice(MM))**

Fig. 12. Additional experimental results with CO dataset

being the use of different verify operator functions, as specified in A.2.

Q4: GNN *explanation containing specific structures*. The query Q4 leverages SPARQL to filter explanations that contain specific structural pattern P. For example, it can be used to identify a fraud triangle, where three "fraud" nodes are mutually connected through fraud-to-fraud edges, potentially indicating a small, tightly-knit fraud group engaged in coordinated malicious activities. Specifically, Let $\mathcal{V}$ be an infinite set of variables, C be the set of class labels of the GNN classification task, and R be the edge labels, if specified. A SliceGXQ pattern P is a fragment of SPARQL pattern that refines the returned explanations via pattern matching, which is an expression generated from the following grammar [40]:

$$P := t | P_1 \wedge P_2 | \ P \ \cup \ P | \ \emptyset$$

Here, P_1 and P_2 are two sub-patterns, and t is a *triple pattern*, an element of $(C \cup \mathcal{V}) \times R \times (C \cup \mathcal{V})$. The grammar allows for the construction of complex patterns, such as a fraud triangle, by combining simpler sub-patterns through logical operations, such as conjunction and disjunction. For SliceGXQ query Q4, a query parser first parses it into a query plan that contains two sub-queries Q_1 and Q_2, to be processed sequentially. (1) Sub-query Q_1 is an explanatory query that computes $\mathcal{G}_s$, following a configuration $\mathcal{C}$ readily derived from the SliceGXQ statement, by invoking SliceGX explanation generation algorithms (Section V). (2) The second sub-query Q_2 is a regular "Construct" SPARQL query that retains the explanations which match the specified SliceGXQ patterns, if any. This step can be efficiently supported by existing SPARQL graph engines [3].

Q5: Sort GNN *explanation*. This query facilitates organizing the returned explanations by sorting them based on node IDs of multiple test nodes to be explained in ascending order after explanation generation phrase. The ordered results are particularly useful when dealing with large-scale datasets, as they simplify visualization and analysis by presenting the explanations in a clear and logical sequence.