# Declarative Explanations for Graph Neural Networks: A Demonstration

Haitong Tang
tht@zju.edu.cn
Zhejiang University
Hangzhou, China

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

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

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

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

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

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

## ABSTRACT

Graph Neural Networks (GNNs) have proven effective in graph-learning tasks such as node classification. However, their "black-box" nature complicates understanding their decision-making process. Existing explainability methods primarily focus on explaining the final output of a GNN in a static, one-time manner, leaving a gap in providing *detailed, progressive* insights into how the GNN interacts with data throughout the inference process. Users are typically interested in exploring how GNNs derive classification results through interaction with their internal layers. Furthermore, they may prioritize certain categories of explanations over others based on *domain-specific* requirements. Manually handling all these demands is cumbersome, highlighting the need for a *declarative* wrapper that enables users to customize "explanatory queries", i.e., queries to generate more flexible and specific explanations.

We demonstrate SliceGXQ, an end-to-end declarative-style system that supports SPARQL-like explanatory queries, designed to facilitate interactive, layer-wise GNN explanations. In particular, SliceGXQ highlights **(1)** a feature-rich interactive graphical user interface with configurable options (e.g., GNN model selection, query mode, and explanation settings); **(2)** a lightweight SPARQL syntax parser that allows users to express their query needs for generating and accessing explanations; and **(3)** a layer-specific explanation generation algorithm that delivers in-depth insights tailored to the user's query. Putting them together, SliceGXQ demonstrates its practical utility in real-world applications, particularly in GNN model debugging and diagnosis, offering a powerful tool for understanding the inner workings of GNNs through layer-wise explanations. Our demonstration video is at [1].

## CCS CONCEPTS

• **Computing methodologies → Neural networks**; • **Information systems → Graph-based database models**.

*SIGMOD-Companion '25, June 22–27, 2025, Berlin, Germany*
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

| | |
|---|---|
| $Q_1$ | EXPLAIN (M, $\{v\}$) At 3 FROM $\{G, \{1\}\}$ WHERE 4 |
| $Q_2$ | EXPLAIN (M, $\{v\}$) At 3 FROM $\{G, \phi\}$ WITH diagnose |
| $Q_3$ | EXPLAIN (M, $\{v_1, v_2\}$) At 3 FROM $\{G, \phi\}$ WITH interpret |
| $Q_4$ | EXPLAIN (M, $\{v\}$) At 3 FROM $\{G, \{1, 2\}\}$ WHERE $<'Fraud','-','Fraud'>$ |
| $Q_5$ | EXPLAIN (M, $\{v_1, v_2\}$) At 3 FROM $\{G, \{1, 2\}\}$ ORDER BY node.id ASC |

**Figure 1: Example of SliceGXQ explanatory queries. These five examples modify the variables in the query, providing references for users in subsequent applications (§3-§4).**

## KEYWORDS

Graph neural networks, Explainable AI, declarative explanation

## 1 INTRODUCTION

Graph Neural Networks (GNNs) have demonstrated remarkable success across various tasks, including node classification. By leveraging message passing and node feature aggregation, GNNs effectively learn node representations by integrating graph structural information with node features. This capability has enabled the widespread applications of GNNs in domains such as social network analysis [7], bioinformatics [4], and recommendation systems [6].

Due to complex message passing mechanisms and end-to-end learning, GNNs are considered "black-box", making their decision-making process opaque to users. To address this issue, numerous efforts, e.g., GNNExplainer [8], SubgraphX [10], and XGNN [9], have been developed to identify the essential substructures (nodes/edges/subgraphs) and node features to provide explanations of "why" the GNN model gives such a prediction and "which" graph components have the most impact on such an outcome.

However, existing GNN explanation techniques exhibit several limitations: **(1)** Most of them provide *static, one-time* explanations that focus solely on the final outputs rather than offering *progressive insights* into how a GNN processes information at each layer. This lack of fine-grained interpretability hinders users from understanding the internal reasoning of the model; **(2)** many explainers are themselves "black-box" [5], making it difficult to validate their outputs; and **(3)** users often need to tailor explanations to specific requirements, such as filtering for particular substructures in the input or focusing on certain layers of the model, which typically requires manual intervention and domain expertise. Hence, a declarative wrapper is essential for users to efficiently specify explanatory queries and easily customize explanations according to their needs.

To this end, we introduce a declarative framework that supports SPARQL-like explanatory queries to facilitate layer-wise explanations of GNNs. We showcase some SPARQL queries in Figure 1. A SliceGXQ query $Q$ follows a general structured format:

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

Specifically, the following declarative operators are integral to SliceGXQ queries: **(1) EXPLAIN** specifies the primary objective of the query, which is to generate explanations for a pre-trained GNN model $(\mathcal{M})$ and test nodes of interest $(V_T)$; **(2) AT** allows users to inspect specific layer $(l_t)$ within the GNN to interact with, enabling fine-grained analysis; **(3) FROM** defines the graph dataset source $(G)$ and selected layers (Ł) to generate explanations for model behaviors at $l_t$; **(4) WHERE** applies constraints on the explanation, such as filtering by pattern $(P)$ or limiting the result size $(k)$; and **(5) WITH** specifies the explanation mode (e.g., model diagnosis vs. interpretation, as detailed in §3). Additionally, SliceGXQ supports SQL-like operators, such as ORDER BY, which allows users to sort the generated explanations based on specific criteria.

SliceGXQ addresses critical challenges in GNN explanation, including limited granularity, interactivity, and customization by introducing a declarative paradigm that allows users to access layer-wise explanations through SPARQL-like queries. The key contributions of SliceGXQ are summarized as follows.

- To our best knowledge, SliceGXQ is the first system that provides insights into the decision-making process at each layer of the GNNs, and allows users to specify flexible and configurable explanatory queries tailored to application needs.
- SliceGXQ includes a comprehensive visualization toolkit that facilitates dataset and GNN model selection, real-time explanation generation, and statistical analysis. This interactive interface empowers users to gain deeper insights into both data properties and model behavior.
- We demonstrate the effectiveness of SliceGXQ in providing detailed, layer-by-layer insights for model diagnosis through a real-world application. By visualizing how the model processes information at each layer, users can better understand its internal behaviors and refine the model accordingly.

## 2 SYSTEM OVERVIEW

The workflow of SliceGXQ is given in Figure 2. The system comprises four key front-end modules: Graph Data Selection, GNN Model Selection, Query-Based Explanations, and Visualization Displayer. Among these, the Query-Based Explanations module serves as the central interface for front-end and back-end interactions. The SliceGXQ back-end also consists of four primary modules: Query Interpretation, Model Slicing, Explanation Subgraph Finding, and Verification Operator. These modules process user inputs and configurations received from the front-end, retrieve necessary data from the database, and generate interpretability information for the GNN model, which is then returned to the front-end for visualization and further interaction.

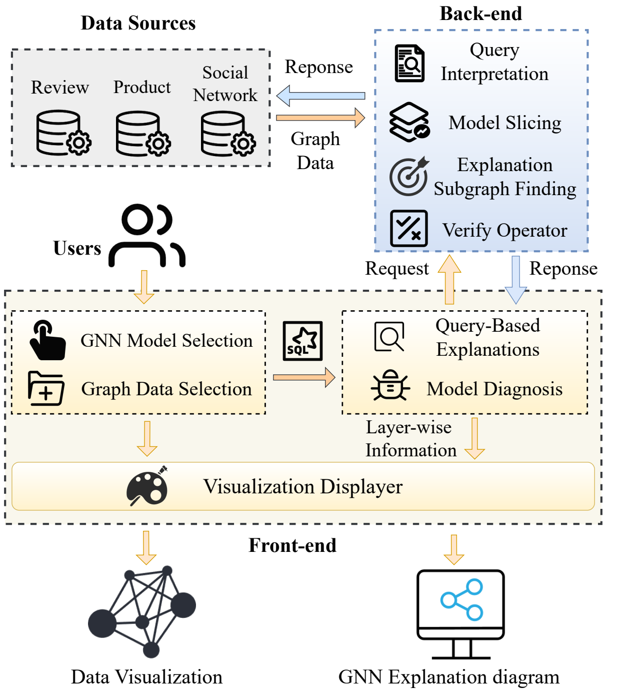


**Figure 2: SliceGXQ comprises a front-end and a back-end. Users can select a dataset and model on the front-end, input a query, and the back-end will execute the corresponding algorithm to generate GNN explanation results.**

**Graph Data and GNN Model Selection:** In the Dataset Selection Module, the user can select the dataset for examination and specify the set of test nodes whose classification results need to be explained. Visualizations of the data are then presented through the Visualization Displayer. In the Model Selection Module, the user can select a GNN and configure its structure, with the corresponding visualizations also displayed. These configurations are forwarded to the Query-Based Explanation module for further processing and subsequent explanation generation.

**Query-Based Explanations:** The core functionality of SliceGXQ lies in its Query-Based Explanations. Users can choose between two input modes depending on their expertise: Configuration Mode, tailored for novice users, and Query Mode, designed for more experienced users. Although Configuration Mode is user-friendly, its functionality is limited. For a more comprehensive and detailed explanation of GNNs, it is recommended to use Query Mode. Query Mode allows for the input of additional parameters and supports a wider range of query patterns, making it suitable for a broader array of scenarios. In Query mode, users input SPARQL-like statements into the query input box within the Query-Based Explanations area. Specific examples of such statements are shown in Figure 1. The SliceGXQ system then parses the input statements and passes them to the back-end, where four modules work sequentially to implement layer-wise, progressive interpretations of the GNN.

**(1) Query Interpretation** module interprets and transforms input SPARQL-like queries into executable tasks for explainability. It enables dynamic exploration of configurations and explanations, providing real-time analysis tailored to user needs. The module ensures queries adhere to the graph schema and are error-free;

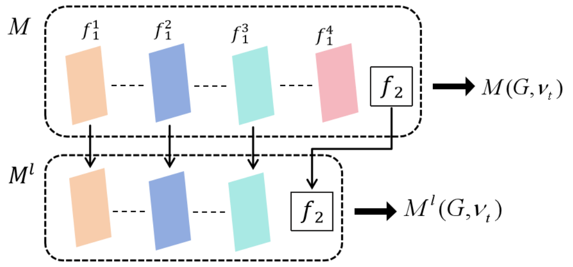


**Figure 3: An example of a sliced model for $l$=3**

otherwise, an error message is returned. Once validated, the query is translated into a configuration $C = (G, \mathcal{M}, V_T, Ł, l_t, k)$, where $G$ represents the input graph, $\mathcal{M}$ denotes the GNN model, $V_T$ is the set of target nodes, Ł refers to a set of source layers in $\mathcal{M}$, $l_t$ is target layer to be explained, and $k$ specifies a size bound w.r.t. the number of explanatory nodes. This configuration is then passed on to the explanation generation algorithm. In "Configuration Mode", user inputs are directly parsed into such a configuration without the need for query statements.

**(2) Model Slicing** module takes the original GNN model as input and segments it into smaller, interpretable local sub-models, each corresponding to a specific layer. These localized sub-models form the foundation for generating layer-specific explanations, allowing for a detailed analysis of the contribution of each layer to the overall model's prediction.

**(3) Explanation Subgraph Finding** employs a greedy strategy to identify high-quality nodes and generates an optimal explanation subgraph for a particular GNN layer. The input includes the original graph, model slices, and configuration parameters, while the output is a subgraph that satisfies explainability and constraint requirements. The purpose is to determine key nodes and their connections to help explain the GNN model's prediction logic.

**(4) Verify Operator** module ensures that the explanation subgraph satisfies both counterfactual and factual properties (defined in §3) for the original GNN model and the sliced model. The input of this module comprises the original graph, the target layer, and the candidate explanation subgraph, while the output is a validated subgraph that clarifies the factors influencing the model's decisions in the target layer. Its key function is to verify the counterfactual and factual relevance of the subgraph to the model's predictions.

**Model Diagnose** module is designed to improve GNN model performance through GNN explanations. It identifies potential weaknesses within a specific layer in a GNN and enhances the model's accuracy and effectiveness. Layer-specific techniques used in this process include **(1)** layer fine-tuning, i.e., refining the parameters of a specific GNN layer while retaining all original layers; **(2)** architecture optimization, that is, prune or simplify the specific layer to optimize the model architecture.

## 3 CORE ALGORITHM

We next introduce the key techniques in SliceGXQ, which aims to generate connected explanation subgraphs at specific layers with flexible configurations. For detailed insights, we refer to our research paper [2]. Given a pre-trained GNN model $\mathcal{M}$, a graph $G$, and a target node $v_t$ with an output $\mathcal{M}(G, v_t)$ to be explained, our explanations consist of a set of *explanatory nodes* that are important for clarifying the output $\mathcal{M}(G, v_t)$ at an "earlier" stage of inference of GNN $\mathcal{M}$ and a *connector set* $V_c$ that ensures the connectivity of the generated subgraph.

**Model Slicing** facilitates finer-grained explanations of GNNs by segmenting a pre-trained GNN model $\mathcal{M}$ into layer blocks, thus enabling layer-by-layer inference-time explanations. Given a GNN $\mathcal{M}$ with a stack of an $L$-layered encoder $f_1$ and a predictor $f_2$, and a selected layer $l \in [1, L]$, the *$l$-sliced model* of $\mathcal{M}$, denoted as $\mathcal{M}^l$, is constructed by **(1)** a feature extractor, comprising the first $l$ GNN layers in encoder $f_1$ that generate output embeddings up to layer $l$; **(2)** the predictor, which applies the model predictor $f_2$ in $\mathcal{M}$ to obtain the classification result through node embeddings. Figure 3 illustrates the construction of an $l$-sliced model, where only the first $l$ layers of the encoder are retained to generate embeddings, followed by the predictor for classification.

**Verify Operator** module ensures that the connected subgraphs adhere to factual and counterfactual properties to both the original GNN model and the sliced model. Specifically, for a target layer $l$, an explanation subgraph $G_s^l$ (a subgraph of $G$) satisfies at least one of the following: **(1)** [Factual] $\mathcal{M}(G, v_t) = \mathcal{M}(G_s^l, v_t)$ and $\mathcal{M}(G, v_t) = \mathcal{M}^l(G, v_t) = \mathcal{M}^l(G_s^l, v_t)$; **(2)** [Counterfactual] $\mathcal{M}(G_s, v_t) = \mathcal{M}(G \setminus G_s^l, v_t)$ and $\mathcal{M}(G, v_t) = \mathcal{M}^l(G, v_t) \neq \mathcal{M}^l(G \setminus G_s^l, v_t)$.

**Explanation Generation** employs a greedy strategy to select high-quality nodes to construct an optimal explanation subgraph for $\mathcal{M}(G, v_t)$ at each specific layer. To evaluate the quality of a subgraph, we introduce a bi-criteria explainability score that combines *relative influence* and *embedding diversity*, prioritizing explanations that accumulate greater influence and reveal more diverse nodes in terms of node embedding. Given a pre-trained GNN model $\mathcal{M}$ and a target layer $l$, the *explainability* of an explanation subgraph $G_s^l$, with the explanatory node set $V_s^l$, at layer $l$ is defined as:

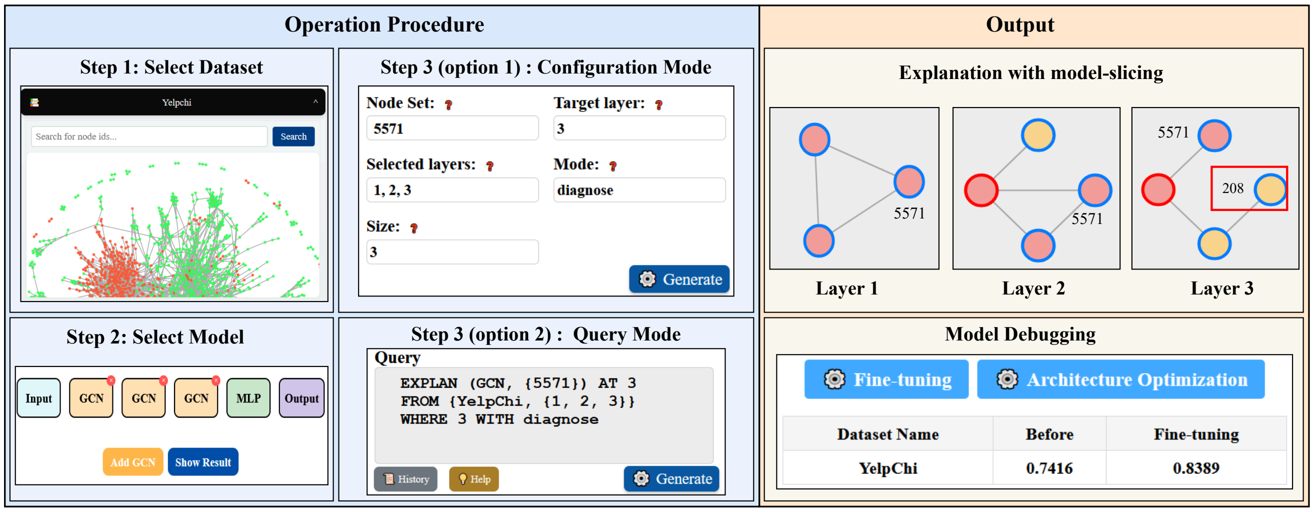
$$f(G_s^l) = \gamma I(V_s^l) + (1 - \gamma) D(V_s^l) \tag{1}$$

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 $f(G_s^l) \in [0, 1]$. While alternative measures exist, they are often tailored to final-layer predictions and overlook the distinct characteristics of intermediate representations in GNNs. To capture the model's behavior at each layer, our bi-criteria explainability score is specifically designed to address these challenges by focusing on intermediate-layer representations, which enables a more nuanced understanding of how intermediate features contribute to the model's decision-making process.

Among all possible explanation subgraphs, an optimal explanation subgraph is favored to **(1)** ensure connectivity to preserve higher-order information; **(2)** maximize the bi-criteria explainability score for the induced subgraph; **(3)** pass the verify operator to ensure factual and counterfactual properties for both the sliced model and original model; and **(4)** satisfy the size constraint on the number of explanatory nodes in the configuration parameters.

This process requires generating all explanation subgraphs sequentially for each test node and each GNN layer. To deal with multiple test nodes or deep GNN models, we introduce two optimization techniques for incrementally generating explanations [2].

**Diagnose vs. Interpret Mode** extends explanations to support two distinct scenarios, each targeting different objectives in understanding and analyzing GNNs: **(1) Diagnose Mode** focuses on



Node Set: 5571
Target layer: 3
Selected layers: 1, 2, 3
Mode: diagnose
Size: 3

Query
```
EXPLAN (GCN, {5571}) AT 3
FROM {YelpChi, {1, 2, 3}}
WHERE 3 WITH diagnose
```

| Dataset Name | Before | Fine-tuning |
|---|---|---|
| YelpChi | 0.7416 | 0.8389 |

**Figure 4: The** SliceGXQ **system consists of three parts: the dataset module, the model module, and** SliceGXQ **inference module. Each module includes user settings and visualization.**

identifying the key components responsible for "incorrect" results in a specific layer when misclassification occurs in model $\mathcal{M}$; and **(2) Interpret Mode** aims to understand the internal inference process of GNNs. We extend the notion of explanation with model-slicing to these two scenarios, with their applications in [2].

## 4 DEMONSTRATION SCENARIO

In this section, we present the workflow of SliceGXQ and demonstrate its advantages through two critical applications: model diagnosis and model debugging. These scenarios highlight the practical value and versatility of SliceGXQ in addressing real-world challenges and optimizing system performance. SliceGXQ code at [3].

**Workflow:** SliceGXQ is designed to facilitate a streamlined, three-step process to enhance user experience and operational efficiency. **Step 1: Dataset Selection** - Users begin by choosing a dataset from the drop-down list, as shown in the upper left corner of Figure 4. They can then visualize the types and quantities of nodes and edges using pie and bar charts, respectively. Nodes of interest can be added to the collection by specifying their identifiers or directly clicking on the graph. **Step 2: Model Selection** - Users configure GNN architectures by adding or removing layers with different functionalities and can view prediction results in a paginated tabular format. **Step 3: SliceGXQ Query** - Users can choose between Configuration Mode and Query Mode for input. SliceGXQ processes the input by sending it to the back-end for inference. Configuration Mode is designed for beginners unfamiliar with SliceGXQ, while Query Mode is tailored for experts proficient in SliceGXQ. For guidance on query syntax, users can click "Help" to auto-generate a template in the input box.

**Diagnosis Scenario:** To demonstrate the functionality of SliceGXQ, consider an example in which node 5571 (with ground truth label "Benign") is misclassified as "Fraud" by GCN $\mathcal{M}$. A user seeks to understand why and when (at which layer) the error occurs. By inputting a query statement in the Query input box, as shown in Figure 4, SliceGXQ processes the query by passing the parameters to the back-end, where the system sequentially invokes Query Interpretation, Model Slicing, Explanation Subgraph Finding, and Verify Operator modules to generate explanations. The GNN explanation results are then transmitted to the front-end, where they are visually presented in a layer-wise manner. The explanations at layers 1 to 3 demonstrate that $\mathcal{M}$ remains accurate up layer 2, suggesting that the error likely originates at hop 3, where node 208 (labeled "Fraud") introduces bias into the inference process.

**Model Debugging Scenario:** Model debugging is facilitated through two complementary approaches, each guided by SliceGXQto address deficiencies in GNN performance. The first approach, debugging via *fine-tuning*, focuses on refining a specific GNN layer identified by SliceGXQ as contributing most to classification errors, while retaining all other layers unchanged. The second approach, debugging via *architecture optimization*, simplifies the model by pruning the layer highlighted by SliceGXQ as responsible for classification errors or over-smoothing. For example, on the YelpChi dataset, a fine-tuned 2-layer GNN outperforms the original 3-layer model by 0.8% in accuracy, as shown in Figure 4. These results underscore the effectiveness of SliceGX-driven debugging strategies in improving both model performance and efficiency through precise fine-tuning and structural optimization.

## REFERENCES

[1] 2025. https://vimeo.com/1052174431.
[2] 2025. https://anonymous.4open.science/r/SliceGX/Research_paper.pdf.
[3] 2025. https://github.com/Hai0709/SliceGXQ.git.
[4] T. Liu, Y. Wang, R. Ying, and H. Zhao. 2024. MuSe-GNN: Learning unified gene representation from multimodal biological graph data. *NeurIPS* (2024).
[5] C. Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.* 1, 5 (2019), 206–215.
[6] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui. 2022. Graph neural networks in recommender systems: a survey. *Comput. Surveys* 55, 5 (2022), 1–37.
[7] L. Yang, Z. Liu, Y. Dou, J. Ma, and P. S. Yu. 2021. Consisrec: Enhancing GNN for social recommendation via consistent neighbor aggregation. In *SIGIR*.
[8] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec. 2019. Gnnexplainer: Generating explanations for graph neural networks. In *NeurIPS*.
[9] H. Yuan, J. Tang, X. Hu, and S. Ji. 2020. XGNN: Towards model-level explanations of graph neural networks. In *KDD*.
[10] H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji. 2021. On explainability of graph neural networks via subgraph explorations. In *ICML*.