ProgNet: Program-Grounded Evidence Composition for Interpretable Graph Classification

Minseok Jeon
Daegu Gyeongbuk Institute of Science and Technology
Daegu, Republic of Korea
minseok_jeon@dgist.ac.kr

Seunghyun Park
LG CNS
Seoul, Republic of Korea
s.hyun@lgcns.com

Jun-Gi Jang*
Daegu Gyeongbuk Institute of Science and Technology
Daegu, Republic of Korea
jungi@dgist.ac.kr

Abstract

We present ProgNet, a graph learning framework for interpretable graph classification that treats explanatory structures as first-class, reusable components of the prediction mechanism. Departing from existing methods that generate isolated, instance-specific explanations, ProgNet introduces a paradigm where reasoning is grounded in a shared vocabulary of reusable structural programs. Specifically, ProgNet represents each graph using a shared vocabulary of human-interpretable programs written in a graph pattern description language, grounding predictions in explicit structural evidence rather than latent embeddings alone. The vocabulary is constructed to promote both coverage and diversity, yielding compact and reusable structural primitives that generalize across instances. Classification is performed via an inherently decomposable evidence composition network that scores and aggregates program-level evidence, resulting in predictions whose logits admit additive, signed attributions. Extensive experiments on eight graph classification benchmarks demonstrate that ProgNet achieves competitive predictive accuracy while providing more faithful explanations.

CCS Concepts

• **Computing methodologies** → **Neural networks**; • **Information systems** → **Data mining**.

Keywords

graph classification, interpretable graph learning, graph neural networks, graph pattern mining

ACM Reference Format:
Minseok Jeon, Seunghyun Park, and Jun-Gi Jang. 2026. ProgNet: Program-Grounded Evidence Composition for Interpretable Graph Classification. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '26), August 09–13, 2026, Jeju Island, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3770855.3817844

1 Introduction

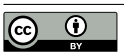
Graphs are a fundamental data structure that can naturally represent complex relationships and interactions in diverse real-world data, ranging from molecular structures and social networks to program execution traces and knowledge bases. This versatility has led to the widespread adoption of graph classification across various decision-critical domains such as drug discovery [14, 16, 23, 27]. In such domains, stakeholders require both accurate predictions and clear explanations.

Recently, graph neural networks (GNNs) [26] have emerged as the dominant approach for graph classification. GNNs demonstrate strong potential through their ability to learn effective graph representations by iteratively aggregating information from neighboring nodes. This capability enables GNNs to capture complex structural patterns and achieve high accuracy on various graph classification tasks. As a result, GNNs are now widely adopted in real-world applications across diverse domains [33]. However, despite their accuracy, standard GNNs often operate as black boxes: the evidence driving a prediction remains entangled in continuous embeddings, making it difficult to explicitly identify *what* structural patterns are responsible for the decision. This limitation becomes critical in domains where understanding the underlying reasoning process is as crucial as achieving high predictive performance.

To address this issue, many explainable and interpretable GNN techniques have been proposed [11]. However, existing approaches often struggle to provide faithful explanations, and they frequently exhibit a trade-off between predictive accuracy and explanation quality. Specifically, it remains challenging to connect GNN embeddings to discrete structural patterns for intuitive and faithful explanations. As illustrated in Figure 1, methods that preserve the accuracy of strong GNN encoders (e.g., GNN+SubgraphX [32] and GSAT [19]) tend to provide weak or unstable explanations, while inherently interpretable approaches (e.g., PL4XGL [10]) often sacrifice prediction accuracy. These recurring limitations highlight the need for an alternative approach that achieves both high predictive accuracy and explainability. A common challenge across existing approaches is that explanatory structures are not treated as shared, reusable components of the prediction process, but are instead generated independently for individual instances. This instance-level isolation prevents models from learning a consistent structural vocabulary across the dataset, often leading to unstable explanations that fail to capture the generalized logic of the task. This motivates a framework in which explanatory structures are treated as first-class, reusable components that drive the prediction process.

To this end, we propose ProgNet, a framework that integrates neural representation learning with an explicit, program-level evidence space for transparent graph classification. ProgNet represents each graph through a shared vocabulary of human-interpretable programs written in a graph pattern description language, enabling

*Corresponding author.



KDD '26, Jeju Island, Republic of Korea

ACM ISBN 979-8-4007-2259-2/2026/08
https://doi.org/10.1145/3770855.3817844

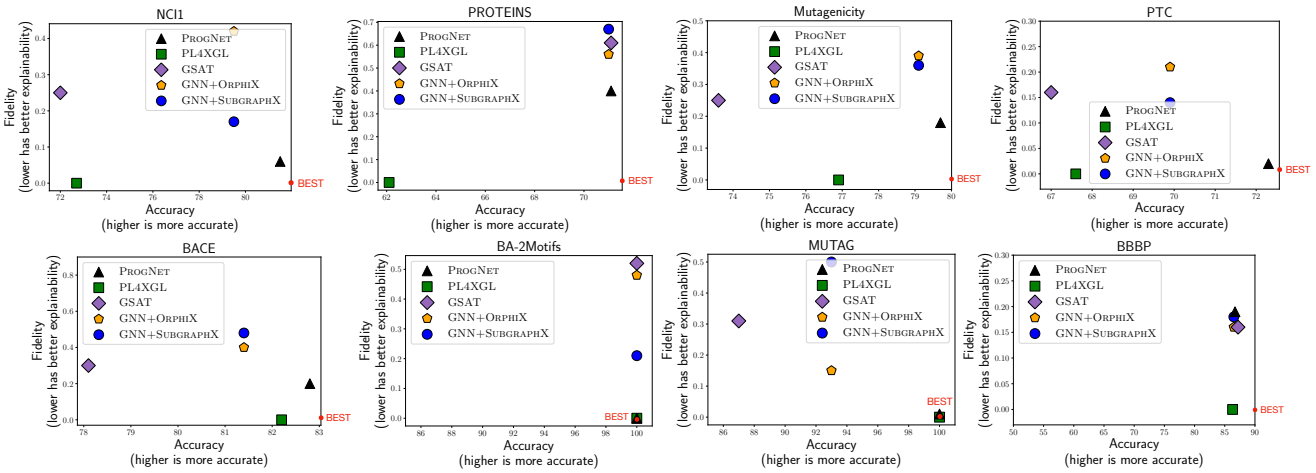


Figure 1: Accuracy-fidelity balance across the eight benchmarks. For the x-axis, higher values indicate better accuracy. For the y-axis, lower values indicate better fidelity (better explainability).

explicit reasoning beyond latent features. We construct a compact vocabulary by selecting diverse structural patterns (i.e., programs) that maximize coverage while minimizing redundancy. Finally, we propose a decomposable evidence composition network that scores program relevance via bilinear interactions and aggregates evidence through additive composition. This design yields intrinsic interpretability: predicted logits decompose into signed program-level contributions for the predictions. The predictions can be explained through the programs (i.e., graph patterns) with high contributions. The experimental results demonstrate that ProgNet achieves both high accuracy and interpretability. ProgNet attains competitive accuracy across representative graph classification benchmarks while providing high-quality explanations for the predictions compared to various existing baselines (e.g., Figure 1). In addition, ProgNet provides dataset-level (i.e., generalizable) evidence that can be reused across the dataset, which is not the case for existing methods that provide instance-level evidence.

We summarize our contributions as follows:

- **Program-Grounded Evidence Composition.** We propose an inherently decomposable architecture that integrates neural representation learning with an explicit, program-level evidence space, thereby enabling predictions to be directly attributed to a set of human-interpretable structural primitives.
- **Diversity-Preserving Structural Primitive Discovery.** We introduce a diversity-preserving vocabulary construction strategy that identifies reusable structural primitives across the dataset. By curating a compact vocabulary of program-level evidence, ProgNet enables the transition from instance-specific explanations to generalizable, dataset-level structural insights.
- **Empirical Validation.** Extensive experiments on eight representative graph classification benchmarks demonstrate that ProgNet achieves competitive accuracy while consistently providing more faithful explanations compared to state-of-the-art baselines. Our source code is publicly available.[1]

[1]https://github.com/dgistpl/ProgNet

2 Related Work

Graph Neural Networks. Graph Neural Networks (GNNs) are the dominant methods in graph machine learning due to their high accuracy. In graph classification, GNNs [3, 6] perform a message-passing procedure to generate node and edge representations that capture local structural information (i.e., neighborhood features). For instance, GCN [12] simplifies spectral graph convolution by aggregating information from local neighborhoods. GAT [25] learns to weigh the importance of neighboring nodes via attention mechanisms, enabling more relevant neighbors to contribute more when updating a node's embedding. GIN [28] aggregates neighbor features using a sum operation and applies an MLP. After message passing, GNNs apply a pooling mechanism to aggregate node-level information into a single graph-level representation [5]. The graph is then classified using this aggregated embedding. However, because GNNs make predictions based on these opaque vectorized representations, it is challenging to explain their predictions [11]. This lack of interpretability limits the usage of GNNs in decision-critical domains that require both accurate predictions and clear explanations.

Explainable Graph Machine Learning. To develop explainable and accurate graph machine learning methods, various approaches have been proposed. A mainstream approach is to develop post-hoc explanation methods [17, 29, 32] that provide explanations after GNNs make predictions. Given a graph with its prediction, for instance, a state-of-the-art post-hoc explanation method SubgraphX [32] generates a subgraph that is considered to be the reason behind the prediction. Other post-hoc techniques such as GraphChef [20] and GraphLime [8] generate interpretable surrogate models that mimic the GNN's behavior. Another line of work focuses on rationalization-based approaches [13] that identify important parts of the input graph during the prediction process. For instance, GSAT [19], an intrinsically interpretable approach based on stochastic attention, identifies task-relevant subgraphs through the information bottleneck principle and provides them as rationales. To develop inherently interpretable graph machine learning, recent work has developed symbolic (i.e., non-neural)

graph machine learning methods. For instance, PL4XGL [10] mines graph patterns described as GDL programs from training data and uses them in graph classification.

Graph Pattern Description Languages. To explicitly describe meaningful graph patterns, various graph pattern description languages have been designed. The most traditional graph pattern description method is the subgraph representation. Beyond explaining GNNs [29], subgraphs have served as a pattern description language in graph pattern mining [1, 9]. As a pattern description language, however, subgraphs have limited expressiveness, often failing to capture key patterns in real-world datasets [10]. To address this limitation, more expressive graph pattern description languages have been explored in the programming language community [18, 21, 24]. For instance, Cypher [4, 7] is a representative graph pattern description language that uses constraints (rather than specific values) to describe meaningful graph patterns and has been widely adopted for querying graph databases. GDL (graph description language) [10] is a relatively recent one that uses intervals to describe feature value ranges in graph patterns.

3 Proposed Framework

Now, we present our approach ProgNet in detail. We first introduce the problem and challenges. Then, we illustrate our idea and how ProgNet addresses the problem.

3.1 Problem Formulation

Let $\mathcal{G} = \{G_1, \ldots, G_m\}$ be a set of graphs, where each graph G_i is associated with a label $y_i \in \{1, \ldots, C\}$. In a graph, each node (resp., edge) is associated with a d-dimensional (resp., e-dimensional) feature vector. Our objective is to learn a classifier $f : \mathcal{G} \rightarrow \{1, \ldots, C\}$ that minimizes graph classification error while providing *intrinsic* and *faithful* explanations.

3.2 Challenges & Ideas

To address the problem, we tackle the following three challenges.

C1. Absence of an explicit, shared evidence interface. Standard GNNs operate in latent embedding spaces, and many explainers provide isolated, instance-specific rationales. This makes it difficult to expose *reusable* and *inspectable* evidence units that are consistent across the dataset.

C2. Redundant and low-diversity structural primitives. The space of graph substructures is combinatorially large. Naïve strategies often collect redundant patterns, limiting coverage and diversity.

C3. Reliable composition under soft grounding. Scoring patterns in a differentiable manner while preserving decomposable aggregation for faithful attribution remains challenging.

To address these challenges, we propose the following ideas:

I1. Declarative program-based evidence interface. We represent each graph using a shared vocabulary of human-interpretable programs, enabling reusable and inspectable evidence across instances.

I2. Diversity-preserving evidence discovery via precision–coverage greedy selection. We construct a compact yet informative program vocabulary by employing a greedy selection that maximizes coverage while reducing redundancy.

I3. Compositional and faithful prediction via decomposable evidence aggregation. We design an evidence-aware reasoning network that performs context-aware soft scoring of programs and composes their signed contributions additively, enabling transparent multi-evidence reasoning and faithful program-level attribution.

Given a training set, ProgNet first extracts candidate programs from training graphs and selects a compact, representative subset to form the global evidence vocabulary. Given an input graph, ProgNet encodes it with a GNN and computes program relevance scores using the decomposable reasoning network. The final prediction is obtained by composing program-level attributions.

3.3 Declarative Evidence Interface

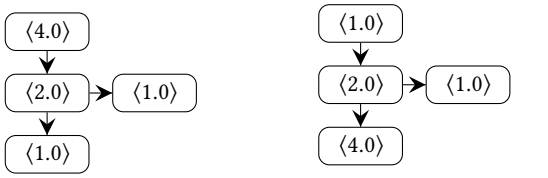
To bridge the gap between low-level graph structures and high-level symbolic reasoning, we introduce a *declarative evidence interface* for graph classification. For example, in molecular graphs, low-level evidence corresponds to raw primitives such as atoms and their local feature vectors, which are difficult to interpret in isolation. High-level evidence, in contrast, captures chemically meaningful concepts (e.g., functional groups) expressed as logical programs that compose multiple primitives into human-readable rules. This interface specifies a simple contract: a graph G is mapped to an explicit set of evidence activations by evaluating a collection of GDL programs. Each program P serves as a human-readable logical unit that captures graph patterns. By exposing these program-level signals as first-class components of the prediction pipeline, our model provides compositional and faithful explanations.

GDL is a lightweight declarative programming language for specifying human-interpretable graph patterns. A GDL program P consists of logical node and edge variables associated with feature value constraints. Below, we use the notation $\overline{A}$ to denote a sequence of elements in A.

Syntax. A GDL program $P = (\overline{\delta}_V, \overline{\delta}_E)$ consists of node descriptions $\overline{\delta}_V$ and edge descriptions $\overline{\delta}_E$. A node description δ_V is a pair of a node variable and a feature value constraint (e.g., `node` x $<\overline{\phi}>$). A feature value constraint ($<\overline{\phi}> = <\phi_1, \ldots, \phi_d>$) is a d-dimensional vector of intervals (e.g., $<[0.0, 2.0], \ldots, [3.0, 5.0]>$) where the i-th interval ϕ_i constrains the i-th value of the feature vector. An edge description δ_E is a pair of an edge variable and a feature value constraint (e.g., `edge` $x <\phi_1, \ldots, \phi_e>$).

Semantics. A GDL program describes a set of graphs that satisfy the program. A graph $G_i = (V_i, E_i)$ satisfies a program P (denoted $G_i \models P$) if there exists a mapping that assigns a distinct node to each variable such that the feature values of the corresponding nodes and edges satisfy the feature value intervals. That is, if there exists a subgraph $G_i' \subseteq G_i$ that satisfies all structural and feature value constraints of P, then $G_i \models P$. We define the semantics of P as $[\![P]\!] = \{G \mid G \models P\}$, the set of all graphs satisfying P.

Example. Figure 2 is an example to illustrate how GDL programs work. Figure 2a and 2b are two example graphs where each node is



(a) Graph G_1 **(b) Graph G_2**

```
node x <[3.0, 4.0]>
node y <[2.0, 2.0]>
node z <[1.0, 1.0]>
edge (x, y)
edge (y, z)
```

(c) GDL program P_1

```
node x <[1.0, 1.0]>
node y <[0.0, 5.0]>
node z <[1.0, 1.0]>
edge (x, y)
edge (y, z)
```

(d) GDL program P_2

Figure 2: Example graphs and GDL programs

associated with a one-dimensional feature vector. Figure 2c and 2d are two example GDL programs. P_1 describes a length-2 path where nodes (x, y, z) must satisfy feature intervals $\langle[3.0, 4.0]\rangle$, $\langle[2.0, 2.0]\rangle$, and $\langle[1.0, 1.0]\rangle$, respectively. Accordingly, G_1 satisfies P_1 ($G_1 \models P_1$) due to a subgraph ⟨4.0⟩→⟨2.0⟩→⟨1.0⟩, whereas G_2 does not satisfy P_1 ($G_2 \notin [\![P_1]\!]$) since it contains no subgraph satisfying P_1. Meanwhile, G_2 satisfies P_2 ($G_2 \in [\![P_2]\!]$) because of a subgraph ⟨1.0⟩→⟨2.0⟩→⟨1.0⟩, whereas G_1 does not ($G_1 \notin [\![P_2]\!]$). Compared to rigid subgraph templates, GDL provides a more expressive and vocabulary-friendly pattern language by using feature value ranges in addition to topology, while still capturing fixed subgraphs as a special case.

A natural alternative for building an evidence vocabulary is to employ subgraphs with specific feature values. However, such a representation is inherently instance-specific, which conflicts with our goal of constructing a *shared* evidence space that supports *dataset-level* reuse and compositional reasoning. This motivates our use of declarative programs as discrete evidence units.

We now describe how to extract GDL programs from graphs. Given a graph G_i from a training set $\mathcal{D} = \{(G_i, y_i)\}_{i=1}^m$, our interface extracts a single GDL program $P_i = \textsc{Extract}(G_i, \mathcal{D})$ that describes a graph pattern in G_i (i.e., $G_i \models P_i$). Repeating this extraction over the training set yields a collection of per-graph programs $\mathcal{P} = \{P_i\}_{i=1}^m$, from which we construct a compact vocabulary $\mathcal{Q}$ in Section 3.4. In addition, we define a function l that maps each program P_i to the class label of the source graph, i.e., $l(P_i) = y_i$.

For candidate program generation, we follow the GDL formalism and program-mining procedure of [10], using it as a candidate generator for constructing ProgNet's evidence vocabulary. This choice provides an established pipeline for obtaining graph patterns that align with the requirements of our declarative interface. Building upon this interface design, we focus the remainder of our framework on two critical challenges: (i) constructing a compact, diversity-preserving vocabulary from the large and redundant candidate programs, and (ii) integrating the selected programs as first-class symbolic evidence for GNN-based prediction and attribution.

Algorithm 1 Vocabulary Construction

Require: Training set $\mathcal{D} = \{(G_i, y_i)\}_{i=1}^m$, Target vocabulary size n
Ensure: Program Vocabulary $\mathcal{Q} = \{P_1, P_2, \ldots, P_n\}$

1: **procedure** Construct($\mathcal{D}$)
2: $\quad\mathcal{P} \leftarrow \bigcup_{(G_i, y_i) \in \mathcal{D}} \textsc{Extract}(G_i, \mathcal{D})$
3: $\quad\mathcal{Q} \leftarrow \emptyset$
4: $\quad$**while** $|\mathcal{Q}| < n$ **do**
5: $\quad\quad P \leftarrow \operatorname{argmax}_{P \in \mathcal{P}} \frac{\left|((([\![P]\!] \cap \mathcal{G}_{tr}) \setminus C(\mathcal{Q})) \cap \mathcal{G}_{y(P)}\right|}{|\mathcal{G}_{tr} \cap [\![P]\!]|}$ ▷ Eq. (1)
6: $\quad\quad\mathcal{Q} \leftarrow \mathcal{Q} \cup \{P\}$
7: $\quad$**return** $\mathcal{Q}$

3.4 Vocabulary Construction

The first challenge is to curate a compact yet representative vocabulary from the vast space of programs $\mathcal{P} = \{P_i\}_{i=1}^m$. A naive approach is to use all the extracted programs. However, this leads to an excessively large evidence space filled with *redundant* and overlapping patterns. This not only increases computational overhead but also obscures the clarity of explanations. An alternative is to select the top-k programs based on precision scores. Each program P_i in $\mathcal{P}$ is scored by how precisely it describes its associated label $l(P_i)$ on the training set $\mathcal{D}$: $\frac{|\mathcal{G}_{l(P_i)} \cap [\![P_i]\!]|}{|\mathcal{G}_{tr} \cap [\![P_i]\!]|}$, where $\mathcal{G}_{tr}$ denotes the set of training graphs (i.e., $\mathcal{G}_{tr} = \{G_j \mid (G_j, y_j) \in \mathcal{D}\}$) and $\mathcal{G}_{l(P_i)}$ denotes the graphs with label $l(P_i)$ in the training set (i.e., $\mathcal{G}_{l(P_i)} = \{G_j \mid (G_j, y_j) \in \mathcal{D}, y_j = l(P_i)\}$). However, this strategy often fails to ensure sufficient coverage of the dataset. Since high-precision programs may only describe a small and specific subset of graphs, a top-k list can be dominated by minor variations of the same motif, leaving a large portion of the dataset unexplained.

To address the challenge, we propose a greedy selection strategy that prioritizes marginal precision on the uncovered data. We iteratively construct the vocabulary $\mathcal{Q}$ by selecting a program P in the candidate set $\mathcal{P}$ that maximizes the following objective:

$$\operatorname*{argmax}_{P \in \mathcal{P}} \frac{\left|((([\![P]\!] \cap \mathcal{G}_{tr}) \setminus C(\mathcal{Q})) \cap \mathcal{G}_{l(P)}\right|}{\left|\mathcal{G}_{tr} \cap [\![P]\!]\right|} \tag{1}$$

where we define $C(\mathcal{Q}) = \bigcup_{P' \in \mathcal{Q}} ([\![P']\!])$ as the set of graphs covered by the programs in the vocabulary $\mathcal{Q}$.

The core logic of Eq. (1) lies in balancing predictive reliability, coverage, and diversity. By subtracting $C(\mathcal{Q})$ in the numerator, we measure a program's *label-consistent coverage* specifically on the portion of the training set that remains uncovered. Meanwhile, the denominator $|[\![P]\!] \cap \mathcal{G}_{tr}|$ prefers that the newly added program has little overlap with the already selected programs in $\mathcal{Q}$. For example, suppose two programs P_1 and P_2 satisfy $[\![P_1]\!] \cap \mathcal{G}_{tr} = ([\![P_2]\!] \cap \mathcal{G}_{tr}) \setminus \{G\}$ and $G \in C(\mathcal{Q})$. That is, P_2 additionally includes a training graph G that is already covered by $\mathcal{Q}$ compared to P_1. Then, P_2 will receive a lower score than P_1 as they share the same numerator but P_2 has a larger denominator (larger overlap with $\mathcal{Q}$).

Based on Eq. (1), we construct a compact program vocabulary with high coverage and diversity. The overall procedure for this construction is summarized in Algorithm 1. At line 2, it first generates a set of candidate programs. Then, it iteratively adds a program to the vocabulary $\mathcal{Q}$ until the vocabulary size $|\mathcal{Q}|$ reaches the target size n (lines 4–6). In each iteration, it selects a program P from the candidate set using Eq. (1) and updates the vocabulary $\mathcal{Q}$.

3.5 Decomposable Reasoning Network

The remaining challenge is to harness the expressive power of neural predictors while leveraging the curated program vocabulary $\mathcal{Q}$ for intrinsic interpretability. To operationalize the interaction, we identify three architectural requirements:

(1) **Decomposability**: the final class score admits an exact decomposition into per-program evidence contributions, ensuring intrinsic transparency and enabling faithful attribution.
(2) **Contextuality**: the importance of the same program should be adaptively weighted by the graph's global neural context, so that evidence is interpreted in a data-dependent manner.
(3) **Consistency**: each evidence unit corresponds to a shared program in $\mathcal{Q}$ and is evaluated with a consistent semantics across the dataset, supporting reuse and comparability across instances.

To satisfy these requirements, we propose a decomposable reasoning network that models a tripartite interaction between graph embeddings, program representations, and class prototypes. Formally, let $\mathbf{h}_{G_i} \in \mathbb{R}^D$ be the latent graph representation derived from a GNN encoder, and let $\mathbf{h}_P \in \mathbb{R}^D$ be the learnable embedding for program $P \in \mathcal{Q}$. Let $e_P(G_i) \in \{0, 1\}$ denote the activation indicator of program P on graph G_i (i.e., $e_P(G_i) = 1$ iff $G_i \in [\![P]\!]$). The logit for class c is computed as:

$$f_c(G_i) = \sum_{P \in \mathcal{Q}} e_P(G_i) \cdot \Big(\underbrace{(\mathbf{v}_c^\top \mathbf{h}_P)}_{\text{Class Alignment}} \cdot \underbrace{(\mathbf{h}_{G_i}^\top \mathbf{W} \mathbf{h}_P)}_{\text{Instance Context}} \Big). \tag{2}$$

where $\mathbf{v}_c \in \mathbb{R}^D$ is a class-specific prototype and $\mathbf{W} \in \mathbb{R}^{D \times D}$ is a bilinear interaction matrix. In this architecture, the *Instance Context* term captures **contextuality** by gating a program's contribution based on the neural embedding $\mathbf{h}_{G_i}$. The use of shared program embeddings $\{\mathbf{h}_P\}_{P \in \mathcal{Q}}$, combined with fixed program semantics, ensures **consistency** across the dataset; this facilitates knowledge reuse and enables direct comparability of evidence units across diverse instances. Finally, since $f_c(G_i)$ is defined as an explicit summation over activated programs, the model is **decomposable** by construction, allowing for faithful program-level attribution.

Graph Encoder. To compute the latent graph representations, we employ a standard Graph Convolutional Network (GCN) as the encoder. For a graph $G_i = (V_i, E_i)$ with initial node features $\mathbf{X}_i \in \mathbb{R}^{|V_i| \times d_{in}}$, we first obtain node-level embeddings $\mathbf{H}_i \in \mathbb{R}^{|V_i| \times D}$ via a GCN layer followed by a non-linear activation: $\mathbf{H}_i = \sigma(\text{GCN}(\mathbf{X}_i, E_i))$, where σ is the activation function. The node representations are then aggregated into a fixed-dimensional graph-level embedding $\mathbf{h}_{G_i} \in \mathbb{R}^D$ through global mean pooling: $\mathbf{h}_{G_i} = \frac{1}{|V_i|} \sum_{v \in V_i} \mathbf{h}_{v,i}$, where $\mathbf{h}_{v,i}$ is the v-th row of $\mathbf{H}_i$. While we use a single-layer GCN for simplicity, our framework is model-agnostic and can be integrated with more advanced graph encoders or pooling mechanisms.

Program Embeddings. Each GDL program $P \in \mathcal{Q}$ is represented by a learnable embedding $\mathbf{h}_P \in \mathbb{R}^D$. We collect these embeddings in a matrix $\mathbf{H}_\mathcal{Q} \in \mathbb{R}^{|\mathcal{Q}| \times D}$, which is trained jointly with the GNN encoder under the classification objective. Intuitively, $\mathbf{h}_P$ provides a continuous representation of a discrete symbolic pattern, allowing programs to interact with neural graph representations through the scoring function (e.g., via class–program alignment $\mathbf{v}_c^\top \mathbf{h}_P$ and graph–program context $\mathbf{h}_G^\top \mathbf{W} \mathbf{h}_P$). As a result, the model can leverage the curated vocabulary $\mathcal{Q}$ as first-class evidence while retaining the expressive power of neural features, enabling both accurate prediction and faithful program-level attribution.

Algorithm 2 Explaining ProgNet

Require: graph G_i, trained ProgNet model $\mathcal{M}_\mathcal{Q}$
Ensure: subgraph explanation G_i'
1: **procedure** ExplanationSubgraph(G_i, $\mathcal{M}_\mathcal{Q}$)
2: $G_i' \leftarrow G_i$
3: $\mathcal{Q}' \leftarrow \{P \mid P \in \mathcal{Q}, \alpha(G_i, \hat{c}, P) \geq \tau\}$
4: **repeat**
5: $G_i \leftarrow G_i'$
6: $G_i' \leftarrow \mathit{Refine}(G_i, \mathcal{Q}')$
7: **until** $G_i = G_i'$
8: **return** G_i'

Class Prototypes. For each class $c \in \{1, \ldots, C\}$, we maintain a learnable prototype vector $\mathbf{v}_c \in \mathbb{R}^D$, and collect them into a matrix $\mathbf{V} \in \mathbb{R}^{C \times D}$. These prototypes lie in the same latent space as $\mathbf{h}_G$ and $\mathbf{h}_P$, and directly parameterize the *class alignment* term $(\mathbf{v}_c^\top \mathbf{h}_P)$ in our scoring function. This design allows the model to learn class-specific preferences over programs, thereby emphasizing label-relevant symbolic evidence.

3.6 Explaining Predictions

A key advantage of our architecture is that predictions are explicitly decomposed into per-program contributions. We provide two complementary explanation layers: (i) *program-level attribution* that quantifies how each globally-defined evidence program contributes to the predicted logit for a given instance, and (ii) *instance-grounded evidence extraction* that grounds the top-attributed programs into a compact subgraph of the input graph.

Program-level Attribution over Globally-defined Evidence. For a graph G_i with predicted class $\hat{c}$, we quantify the contribution of each program $P \in \mathcal{Q}$ by its *attribution score*:

$$\alpha(G_i, \hat{c}, P) = e_P(G_i) \cdot (\mathbf{v}_{\hat{c}}^\top \mathbf{h}_P) \cdot (\mathbf{h}_{G_i}^\top \mathbf{W} \mathbf{h}_P), \tag{3}$$

which corresponds to the program-specific term in the predicted-class logit. Programs with large positive attribution scores increase the logit of $\hat{c}$ and thus provide supporting evidence, whereas those with large negative scores act as counter-evidence. Notably, the sign of the final contribution is determined by the directional agreement between the class-alignment term $(\mathbf{v}_{\hat{c}}^\top \mathbf{h}_P)$ and the instance-context term $(\mathbf{h}_{G_i}^\top \mathbf{W} \mathbf{h}_P)$. Specifically, a positive attribution arises when these two terms share the same sign (both positive or both negative), while a negative attribution arises when their signs disagree.

Instance-grounded Evidence Subgraph Extraction. While program-level attribution explains *which* globally-defined programs drive the prediction on G_i, we further ground these programs into a concrete *evidence subgraph*. Algorithm 2 presents our procedure for extracting such a subgraph. The algorithm first identifies important programs $\mathcal{Q}_i'$ based on their attribution scores $\alpha(G_i, \hat{c}, P)$ (line 3). In our algorithm, we select programs with attribution scores larger than a threshold τ, where $\tau > 0$ is a hyperparameter chosen by the user. Then, the algorithm iteratively removes nodes or edges of G_i to find the smallest subgraph $G_i' \subseteq G_i$ that satisfies all programs

in Q_i'. The function $Refine(G_i, Q_i')$ tries to remove nodes or edges such that the refined subgraph G_i' still satisfies all programs in Q_i' (i.e., $\forall P \in Q_i'.\ G_i' \models P$). The procedure terminates when no further node or edge can be removed (i.e., $Refine(G_i, Q_i')$ returns G_i). This yields a compact evidence subgraph G_i' that justifies the model's prediction through the selected important programs Q_i'.

4 Evaluation

In this section, we evaluate the performance of ProgNet on various graph classification benchmarks against several baselines. Our evaluation aims to answer the following research questions:

- **RQ1 (Quantitative Comparison):** How does ProgNet compare to existing graph machine learning methods in terms of accuracy and explainability?
- **RQ2 (Qualitative Analysis):** Can we interpret the predictions of ProgNet through its program-level evidence?
- **RQ3 (Adequacy Analysis):** Does our vocabulary construction procedure produce a compact and diverse vocabulary with high coverage?

Datasets. Our evaluation uses eight widely used graph classification datasets. We use seven molecular datasets: BBBP, BACE, Mutagenicity, PROTEINS, MUTAG, PTC (MR), and NCI1. The labels in these molecular datasets correspond to molecular properties, bioactivity, or toxicity. We note that explainability and accuracy are both important for these molecular datasets, as they are related to drug discovery. Additionally, we include a synthetic dataset, BA-2Motifs, which is widely used for evaluating explainability [32]. Datasets are randomly split into 8:1:1 for training, validation, and test sets.

Baselines. For comparison, we include three types of baseline methods representing different approaches to explainable graph classification. We include traditional graph neural networks (GNNs) with the post-hoc GNN explanation methods SubgraphX [32] and OrphicX [15]. As traditional GNNs, we use GCN [12], GIN [28], and GAT [25]. SubgraphX and OrphicX are state-of-the-art post-hoc GNN explanation methods that generate subgraphs responsible for the predictions. We also include GSAT [19], a rationale-based approach. After a prediction, GSAT provides important edges responsible for the predictions. Lastly, we include PL4XGL [10], which is a symbolic (non-neural) graph learning method that is inherently interpretable. Conceptually, PL4XGL is a decision tree that uses GDL programs as features; the predictions can be interpreted by tracing the decision tree. When training the models, hyperparameters are selected based on the validation set. All experiments are conducted on an AMD Ryzen Threadripper 3990X (64 cores) with an NVIDIA RTX A6000 GPU.

4.1 Quantitative Comparison

Metrics. For comparison, we use the balance between *Accuracy* and *Fidelity*. *Fidelity* [11], which is also known as *Fidelity*$-^{acc}$[31], is a metric that is designed to measure the correctness of the explanations. As interpretability is difficult to compare directly across methods, explainability has been used as a proxy for interpretability. An intuition behind this is that a more interpretable method will provide more correct explanations.

Fidelity assumes that explanations are provided as subgraphs and is defined as follows:

$$Fidelity: \frac{1}{N}\sum_{i=1}^{N}\left(\mathbb{I}(\hat{y}_i = y_i) - \mathbb{I}(\hat{y}_i^{m_i} = y_i)\right)$$

In the above equation, N is the number of explained classifications, y_i represents the original classification result for the i^{th} graph. $\hat{y}_i$ is the classification result for the original graph. m_i denotes the nodes in the explanation subgraph; $\hat{y}_i^{m_i}$ denotes the classification result for the subgraph. The intuition behind *Fidelity* is that the predictions of the model should remain the same when the provided subgraph explanation is given. $\mathbb{I}(a = b)$ is the indicator function that returns 1 if a is the same as b, and 0 otherwise. $\hat{y}_i^{m_i}$ is the prediction of the model when the subgraph explanation is given as an input graph to the model. That is, maintaining the same prediction results in a lower (better) fidelity score. For measuring *Fidelity*, we use Algorithm 2 to generate subgraph explanations from ProgNet's and PL4XGL's program-based predictions.

Following prior work [30], we configured the four methods to generate subgraph explanations of similar size and compare the *Fidelity* score. For all methods except PL4XGL, the size of subgraph explanations can be controlled by hyperparameters; the *Fidelity* score varies depending on the chosen size. Meanwhile, the size of subgraph explanations is chosen by the model itself in PL4XGL; we made other methods generate subgraph explanations of similar size to PL4XGL for fair comparison.

In our evaluation, we do not use other metrics such as the stability and contrastivity [31] as they are unsuitable for our approach. Stability assumes explanations should be the same for similar inputs. In our architecture, however, a mutation that changes whether a graph matches a high-affinity program should yield a different explanation; keeping it the same would itself be misleading. Also, contrastivity assumes that explanations should be different for different labels, but this is also not valid in our method. Our learned vocabulary may contain similar programs associated with different labels. *Fidelity+* is also a well-known metric [31] which measures whether removing the provided subgraph from the graph changes the model's prediction. However, *Fidelity+* is also inappropriate in our architecture: multiple subgraphs may match the same program, so removing one leaves others intact.

Results. Figure 1 shows that ProgNet achieves overall the best balance between accuracy and fidelity for the datasets. In the figure, the x-axis represents the accuracy (higher is better) and the y-axis represents the fidelity (lower is better). The black triangles show the performance of ProgNet. The green squares and purple diamonds show the performance of the baselines PL4XGL and GSAT, respectively. The blue circles and orange pentagons show the performance of the baselines SubgraphX and OrphicX with GNNs, respectively.

Compared to PL4XGL, ProgNet achieves equal or better accuracy for all the datasets. In NCI1, PTC, and Mutagenicity, for example, ProgNet achieves far better accuracy than PL4XGL. This is because PL4XGL is unable to leverage the power of neural networks; ProgNet shows substantially better accuracy than PL4XGL.

Compared to SubgraphX and OrphicX (with GNNs), ProgNet is more accurate and provides overall more faithful explanations. First,

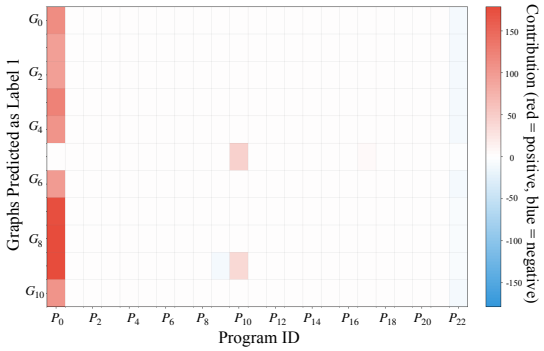


Figure 3: Program contributions to classifying graphs as label 1 in the MUTAG dataset

the baselines are post-hoc techniques that explain after GNNs' predictions; thus, they cannot be more accurate than GNNs. However, ProgNet enhances the accuracy of GNNs by using a high-quality vocabulary that helps predictions. Also, traditional GNNs are considered black-box models; providing faithful explanations is inherently difficult. However, the predictions of ProgNet can be explained by tracing the usage of the vocabulary; ProgNet provides better explanations (i.e., better fidelity) than SubgraphX.

Except for the BBBP dataset, ProgNet achieves overall better performance than GSAT. The main difference between ProgNet and GSAT is that ProgNet uses GDL programs while GSAT uses important subgraphs (e.g., important edges) in the predictions. We note that the graph pattern description language GDL is more expressive than subgraphs used in GSAT: a program is a set of subgraph patterns. This expressiveness gap leads ProgNet to achieve overall better performance than GSAT for the datasets. The detailed accuracy and fidelity comparison is presented in Appendix A.

4.2 Qualitative Analysis of ProgNet

Now, we discuss the interpretability of ProgNet. For illustration, we use interpretation results on the real-world dataset MUTAG.

Dataset-level interpretation. We first examine how the programs in the vocabulary contribute to predictions across the test set. Figure 3 shows how the programs in the vocabulary contribute to classifying the test graphs as label 1 (the molecule is a mutagen). In the dataset, ProgNet classified 11 graphs ($G_0, \ldots, G_{10}$ in Figure 3) into label 1 and the vocabulary $\boldsymbol{Q}$ contains 23 programs ($P_0, \ldots, P_{22}$ in Figure 3). In Figure 3, the x-axis (resp., y-axis) represents the programs in the vocabulary (resp., the test graphs). Each cell depicts the contribution (i.e., $e_P(G_i) \cdot (\mathbf{v}_c^\top \mathbf{h}_P) \cdot (\mathbf{h}_{G_i}^\top \mathbf{W} \mathbf{h}_P)$) of the program to the classification of the graph as label 1. Red cells indicate positive contributions toward label 1, while blue cells indicate negative contributions. As the figure shows, program P_0 serves as dataset-level evidence for label 1: it consistently and predominantly contributes to the classification of 10 out of 11 graphs. That is, understanding the pattern P_0 can lead to a global understanding of the classifications into label 1.

To this end, we can inspect the graph pattern described by P_0. The following is a graphical representation of P_0:

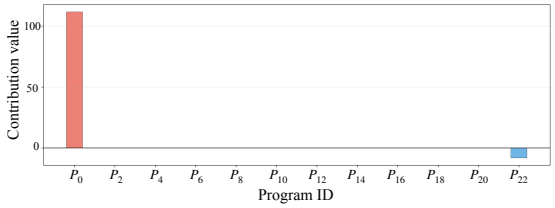


Figure 4: Overall program contribution for classifying G_0

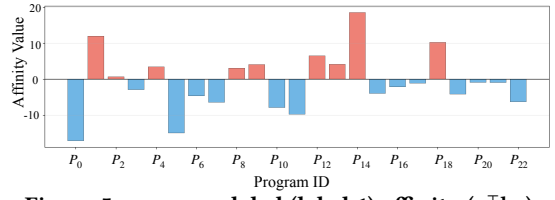


Figure 5: program–label (label 1) affinity ($\mathbf{v}_c^\top \mathbf{h}_P$)

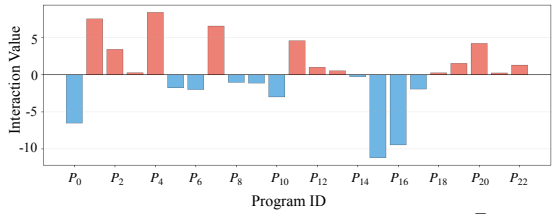
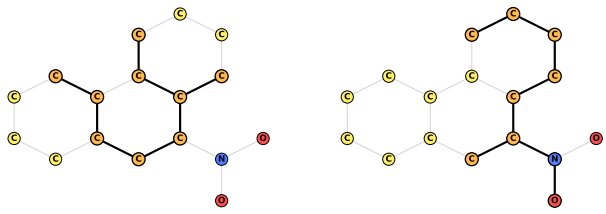
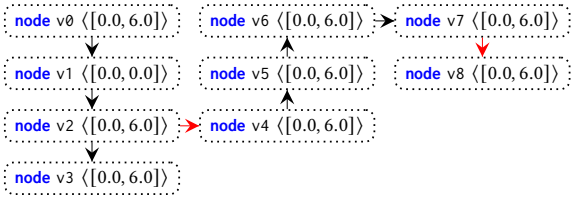


Figure 6: program–graph (G_0) interaction ($\mathbf{h}_{G_i}^\top \mathbf{W} \mathbf{h}_P$)



(a) Matching subgraph 1 **(b) Matching subgraph 2**

Figure 7: Two matching subgraphs of program P_0



For simplicity, we replace edge variables with graphical edges between nodes (node variables). The black-colored edges represent the edge variable with interval $\langle [0, 3] \rangle$ while the red-colored edges represent the edge variable with interval $\langle [0, 0] \rangle$. In the MUTAG dataset, nodes with feature vectors $\langle 0 \rangle$, $\langle 1 \rangle$, ..., and $\langle 6 \rangle$ correspond to carbon (C), nitrogen (N), ..., and bromine (Br), respectively; the node variables with interval $\langle [0, 6] \rangle$ can be matched to any of these atoms. The edge variables with interval $\langle [0, 3] \rangle$ can be instantiated as aromatic, single, double, or triple bonds.

To quantitatively assess the importance of P_0, we also measured the accuracy when P_0 was removed from the vocabulary. We observed that removing the P_0 from the vocabulary drops the accuracy from 1.0 to 0.55, confirming that P_0 indeed captures the essential evidence.

Instance-level interpretation. Next, we zoom into a single graph G_0 to examine how individual programs contribute to its classification. Figure 4 shows that the first program P_0 leads to the classification of the graph as label 1. The remaining programs (except the last one) do not contribute because they are not activated for this graph (i.e., $e_P(G_i) = 0$), meaning G_0 does not match their patterns.

The contribution of each program can be further decomposed into two factors: the program–label affinity ($\mathbf{v}_c^\top \mathbf{h}_P$), shown in Figure 5, and the program–graph interaction ($\mathbf{h}_{G_i}^\top \mathbf{W}\mathbf{h}_P$), shown in Figure 6. From these two figures, we observe that P_0 has a negative affinity with label 1 and a negative interaction with G_0; these two negatives multiply to yield a high positive contribution. This shows that ProgNet's reasoning goes beyond simple rules such as "if a program matches, add a positive weight." Instead, contributions are determined by signed interactions among the program, the input graph, and the target label. This flexibility is actually a strength. A program can contribute positively or negatively to the same label, depending on the sign of its interaction with the input graph.

In the test graph G_0, the concrete subgraphs that match P_0 can also be examined. For example, Figure 7 shows two such subgraphs. In the MUTAG dataset, carbon rings and NO_2 groups are known to be associated with mutagenicity (label 1) [2, 17], and the program P_0 captures parts of both patterns within a single program.

We also verify on the synthetic dataset BA-2Motifs that ProgNet provides high-quality subgraph explanations that exactly capture the motifs corresponding to the labels; we present these results in Appendix B.

4.3 Adequacy of Our Vocabulary Construction

Now, we show that our vocabulary construction procedure achieves high coverage and diversity. To this end, we compare our approach against two alternatives: using all generated programs and using the top-k program selection strategy discussed in Section 3.4.

Coverage Comparison. Figure 8 shows the coverage of each vocabulary construction strategy on the test graphs. The x-axis shows the fraction of selected programs (i.e., $|\mathcal{Q}|/|\mathcal{P}|$), and the y-axis shows the coverage (i.e., the fraction of test graphs satisfied by at least one program in the vocabulary). The blue circles represent the "All" strategy (using all candidate programs), denoting the maximum possible coverage at $x = 1.0$. The red triangles and green squares represent the "Top-k" strategy and our approach, respectively.

As the plots show, our vocabulary achieves high coverage with a compact size. Across most datasets (except PTC), our approach matches the "All" strategy's coverage using only 20% or fewer of the candidate programs. In our approach, the main bottleneck is the program mining process, and the result suggests ProgNet can reduce the program mining cost by about 80% if we mine only the actually collected candidate programs. We discuss the detailed analysis of cost in Appendix C. Compared to "Top-k", our vocabulary achieves significantly higher coverage. For example, on Mutagenicity, our vocabulary achieves 0.99 coverage while "Top-k" achieves only 0.38.

Diversity Comparison. We measure vocabulary diversity using pairwise Jaccard diversity. For each program P in a vocabulary $\mathcal{Q}$, let M_P denote the set of test graphs that satisfy P, i.e., $M_P = [\![P]\!] \cap \mathcal{G}_{te}$ where $\mathcal{G}_{te}$ is the set of test graphs. For two programs P and P', their Jaccard similarity is defined as

$$J(P, P') = \frac{|M_P \cap M_{P'}|}{|M_P \cup M_{P'}|}. \tag{4}$$

Intuitively, $J(P, P')$ is large when two programs tend to cover the same test graphs. We define the pairwise Jaccard diversity of $\mathcal{Q}$ as the average dissimilarity across all program pairs:

$$\mathrm{Div}(\mathcal{Q}) = 1 - \frac{1}{|\mathcal{Q}|(|\mathcal{Q}|-1)} \sum_{\substack{P,P' \in \mathcal{Q} \\ P \neq P'}} J(P, P'). \tag{5}$$

Higher values of $\mathrm{Div}(\mathcal{Q})$ indicate that programs cover more distinct subsets of test graphs (i.e., lower overlap), and thus the vocabulary is more diverse.

Table 1: Diversity of programs

	PROTEINS	BBBP	MUTAG	BA2Motif
Top-k	0.77	0.64	0.0	0.48
All	0.66	0.81	0.76	0.67
Ours	**0.85**	**0.9**	**0.86**	**0.8**
	Mutagenicity	PTC	BACE	NCI1
Top-k	0.4	0.79	0.71	**0.94**
All	0.82	**0.85**	0.9	**0.94**
Ours	**0.84**	**0.85**	**0.95**	0.88

Table 1 shows the diversity of each vocabulary construction strategy, where the best value is boldfaced. In Table 1, our vocabulary achieves the best diversity except for NCI1. In the MUTAG dataset, for instance, the diversity of the Top-k strategy is 0, meaning that all the programs in the vocabulary cover the same test graphs. However, our strategy achieves the best diversity of 0.86 with the same vocabulary size. This demonstrates that our vocabulary construction strategy is also effective at achieving vocabulary diversity.

4.4 Comparison under Matched Model Capacity

To show that the effectiveness of ProgNet is not due to model capacity, we compare ProgNet against GIN and GCN under matched model capacity. Specifically, we re-train GIN and GCN with the number of parameters matched to ProgNet within ±3% (i.e., $100 \times$ #params(GCN or GIN)/#params(ProgNet) $\in [97\%, 103\%]$).

Table 2 reports the accuracy of GIN, GCN, and ProgNet under matched model capacity. As the table shows, ProgNet still achieves overall the best accuracy. This indicates that the accuracy gain of ProgNet is not attributable to model capacity, but rather to its program-grounded evidence composition.

4.5 Extension to Multi-Class Classification

ProgNet extends to multi-class settings. To empirically validate this, we additionally evaluate ProgNet on the ENZYMES dataset, a graph classification benchmark with six labels. Table 3 reports the results. ProgNet achieves 64.6% accuracy, which is competitive with the neural network-based baselines (e.g., GCN at 65.9% and GIN at 64.3%) and substantially better than the interpretable baseline PL4XGL (50.0%).

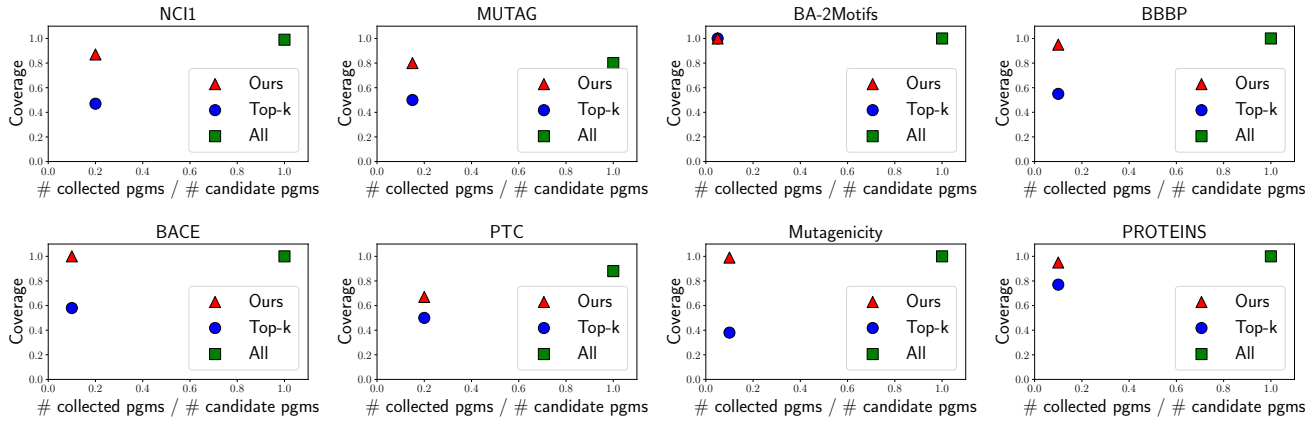


Figure 8: Coverage comparison of the three vocabularies over test graphs. The x-axis presents the portion of programs collected from the candidate programs. The y-axis presents the portion of test graphs that satisfy at least one program in the vocabulary.

Table 2: Accuracy comparison under matched model capacity. The number of parameters of GIN and GCN is matched to that of ProgNet within ±3%.

	NCI1	PTC	MUTAG	BA-2Motifs	PROTEINS	BBBP	BACE	Mutagenicity
GIN	78.4 ± 1.0	64.1 ± 4.5	92.0 ± 3.9	99.4 ± 0.4	58.2 ± 9.6	**86.7 ± 0.6**	79.4 ± 1.4	79.6 ± 0.5
GCN	77.9 ± 0.7	69.9 ± 5.5	90.0 ± 0.0	99.8 ± 0.3	70.9 ± 2.3	83.6 ± 1.9	78.8 ± 1.5	**79.8 ± 0.7**
ProgNet	**81.5 ± 0.8**	**72.3 ± 3.8**	**100.0 ± 0.0**	**100.0 ± 0.0**	**71.1 ± 2.7**	**86.7 ± 1.7**	**82.8 ± 1.4**	79.7 ± 0.5

Table 3: Accuracy comparison on the ENZYMES dataset (six classes).

GIN	GCN	GAT	PL4XGL	GSAT	ProgNet
64.3 ± 3.4	**65.9 ± 3.0**	60.0 ± 5.2	50.0 ± 0.0	54.2 ± 6.5	64.6 ± 4.2

5 Limitations

Though ProgNet achieves competitive accuracy and superior interpretability, it has several limitations.

Vocabulary Construction. ProgNet constructs its vocabulary Q (Section 3.4) prior to, and independently of, the neural training stage. Vocabulary selection is driven solely by the reliability-coverage-diversity objective without direct feedback from the train loss. This could remove programs from the vocabulary that are actually useful for the downstream task, leading to suboptimal performance. Ideally, the vocabulary should be selected in a way that maximizes the downstream task performance. A jointly optimized alternative that selects programs and trains the network end-to-end could recover useful programs that the coverage-only objective filters out. A fully end-to-end joint optimization of vocabulary and network is an interesting direction for future work.

Also, our vocabulary construction (Algorithm 1) is a heuristic greedy procedure. It does not guarantee an optimal solution to the reliability-coverage-diversity objective. Using a more sophisticated selection strategy could potentially achieve better vocabularies.

Preprocessing Cost. The main practical bottleneck of ProgNet is the program-mining stage. For instance, about 270 minutes are required to mine candidate programs in the Mutagenicity dataset. However, as we showed, only a small fraction of the candidate programs are needed to achieve qualified vocabularies; developing a clever mining strategy that only targets this small fraction of programs could substantially reduce the cost.

Interpretability Validation. We do not validate the "human-interpretable" claim through a user study or an expert evaluation. In this paper, we claim that program-level evidence is more human-interpretable than the opaque representations of standard GNNs, and we support this claim through qualitative analyses (Section 4.2 and Appendix B). A formal human-subject evaluation, ideally involving domain experts, would more rigorously substantiate the interpretability claim.

6 Conclusion

In this paper, we presented ProgNet, a declarative program evidence-based framework that addresses the longstanding challenge of achieving both high accuracy and interpretability in graph classification. Our framework represents graphs through a shared vocabulary of human-interpretable graph patterns written in a high-level programming language, enabling predictions to be decomposed into program-level contributions. Extensive experiments demonstrate that ProgNet achieves competitive accuracy compared to state-of-the-art GNNs while consistently providing high-quality explanations. We believe this work opens a promising direction for interpretable graph machine learning that combines the strengths of neural networks and programming languages, with potential applications in drug discovery, molecular property prediction, and other domains where both accuracy and explainability are essential.

Acknowledgments

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2024-00333885 and RS-2026-25490900). This work was supported by the DGIST Start-up Fund Program of the Ministry of Science and ICT (2026010245 and 2026010244).

References

[1] Anna Arpaci-Dusseau, Zixiang Zhou, and Xuhao Chen. 2024. Accurate and Fast Approximate Graph Pattern Mining at Scale. arXiv:2405.03488 [cs.PF] https://arxiv.org/abs/2405.03488

[2] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry* 34, 2 (1991), 786–797. arXiv:https://doi.org/10.1021/jm00106a046 doi:10.1021/jm00106a046

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Barcelona, Spain) *(NIPS'16)*. Curran Associates Inc., Red Hook, NY, USA, 3844–3852.

[4] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An Evolving Query Language for Property Graphs. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1433–1445. doi:10.1145/3183713.3190657

[5] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. 2024. Understanding Pooling in Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 35, 2 (2024), 2708–2718. doi:10.1109/TNNLS.2022.3190922

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf

[7] Yang He, Ruijie Fang, Işıl Dillig, and Yuepeng Wang. 2025. Graphiti: Bridging Graph and Relational Database Queries. *Proc. ACM Program. Lang.* 9, PLDI, Article 216 (June 2025), 25 pages. doi:10.1145/3729319

[8] Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, and Yi Chang. 2022. GraphLIME: Local Interpretable Model Explanations for Graph Neural Networks. *IEEE Transactions on Knowledge and Data Engineering* (2022), 1–6. doi:10.1109/TKDE.2022.3187455

[9] Rana Hussein, Alberto Lerner, Andre Ryser, Lucas David Bürgi, Albert Blarer, and Philippe Cudre-Mauroux. 2023. GraphINC: Graph Pattern Mining at Network Speed. *Proc. ACM Manag. Data* 1, 2, Article 184 (June 2023), 28 pages. doi:10.1145/3589329

[10] Minseok Jeon, Jihyeok Park, and Hakjoo Oh. 2024. PL4XGL: A Programming Language Approach to Explainable Graph Learning. *Proc. ACM Program. Lang.* 8, PLDI, Article 234 (June 2024), 26 pages. doi:10.1145/3656464

[11] Jaykumar Kakkad, Jaspal Jannu, Kartik Sharma, Charu Aggarwal, and Sourav Medya. 2023. A Survey on Explainability of Graph Neural Networks. arXiv:2306.01958 [cs.LG]

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

[13] Tao Lei, Regina Barzilay, and Tommi S. Jaakkola. 2016. Rationalizing Neural Predictions. *CoRR* abs/1606.04155 (2016). arXiv:1606.04155 http://arxiv.org/abs/1606.04155

[14] Yuquan Li, Chang-Yu Hsieh, Ruiqiang Lu, Xiaoqing Gong, Xiaorui Wang, Pengyong Li, Shuo Liu, Yanan Tian, Dejun Jiang, Jiaxian Yan, Qifeng Bai, Huanxiang Liu, Shengyu Zhang, and Xiaojun Yao. 2022. An adaptive graph learning method for automated molecular interactions and properties predictions. *Nature Machine Intelligence* 4, 7 (2022), 645–651. doi:10.1038/s42256-022-00501-8

[15] Wanyu Lin, Hao Lan, Hao Wang, and Baochun Li. 2022. OrphicX: A Causality-Inspired Latent Variable Model for Interpreting Graph Neural Networks. arXiv:2203.15209 [cs.LG] https://arxiv.org/abs/2203.15209

[16] Yunchao "Lance" Liu, Yu Wang, Oanh Vu, Rocco Moretti, Bobby Bodenheimer, Jens Meiler, and Tyler Derr. 2022. Interpretable Chirality-Aware Graph Neural Network for Quantitative Structure Activity Relationship Modeling in Drug Discovery. *bioRxiv* (2022). arXiv:https://www.biorxiv.org/content/early/2022/08/26/2022.08.24.505155.full.pdf doi:10.1101/2022.08.24.505155

[17] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. 2020. Parameterized Explainer for Graph Neural Network. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS'20)*. Curran Associates Inc., Red Hook, NY, USA, Article 1646, 12 pages.

[18] Qiuyang Mang, Jinsheng Ba, Pinjia He, and Manuel Rigger. 2025. Finding Logic Bugs in Graph-processing Systems via Graph-cutting. *Proc. ACM Manag. Data* 3, 3, Article 163 (June 2025), 27 pages. doi:10.1145/3725300

[19] Siqi Miao, Miaoyuan Liu, and Pan Li. 2022. Interpretable and Generalizable Graph Learning via Stochastic Attention Mechanism. arXiv:2201.12987 [cs.LG] https://arxiv.org/abs/2201.12987

[20] Peter Müller, Lukas Faber, Karolis Martinkus, and Roger Wattenhofer. 2023. GraphChef: Learning the Recipe of Your Dataset. In *ICML 3rd Workshop on Interpretable Machine Learning in Healthcare (IMLH)*. https://openreview.net/forum?id=ZgYZH5PFEg

[21] Lunyiu Nie, Shulin Cao, Jiaxin Shi, Jiuding Sun, Qi Tian, Lei Hou, Juanzi Li, and Jidong Zhai. 2022. GraphQ IR: Unifying the Semantic Parsing of Graph Query Languages with One Intermediate Representation. arXiv:2205.12078 [cs.CL] https://arxiv.org/abs/2205.12078

[22] Jinyoung Park, Sungdong Yoo, Jihwan Park, and Hyunwoo J Kim. 2022. Deformable Graph Convolutional Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 7949–7956.

[23] Mengying Sun, Sendong Zhao, Coryandar Gilvary, Olivier Elemento, Jiayu Zhou, and Fei Wang. 2019. Graph convolutional networks for computational drug development and discovery. *Briefings in Bioinformatics* 21, 3 (06 2019), 919–935. arXiv:https://academic.oup.com/bib/article-pdf/21/3/919/33227266/bbz042.pdf doi:10.1093/bib/bbz042

[24] Aditya Thimmaiah, Leonidas Lampropoulos, Christopher Rossbach, and Milos Gligoric. 2024. Object Graph Programming. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 20, 13 pages. doi:10.1145/3597503.3623319

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[26] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24. doi:10.1109/TNNLS.2020.2978386

[27] Jiacheng Xiong, Zhaoping Xiong, Kaixian Chen, Hualiang Jiang, and Mingyue Zheng. 2021. Graph neural networks for automated de novo drug design. *Drug Discovery Today* 26, 6 (2021), 1382–1393. doi:10.1016/j.drudis.2021.02.011

[28] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=ryGs6iA5Km

[29] 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*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/d80b7040b773199015de6d3b4293c8ff-Paper.pdf

[30] Junchi Yu, Jie Cao, and Ran He. 2022. Improving Subgraph Recognition with Variational Graph Information Bottleneck. arXiv:2112.09899 [cs.LG] https://arxiv.org/abs/2112.09899

[31] 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* (2022), 1–19. doi:10.1109/TPAMI.2022.3204236

[32] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. 2021. On Explainability of Graph Neural Networks via Subgraph Explorations. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*. 12241–12252.

[33] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. *CoRR* abs/1812.08434 (2018). arXiv:1812.08434 http://arxiv.org/abs/1812.08434

Appendix

A Accuracy and Fidelity Comparison

Table 4 presents the accuracy comparison results, with the best-performing models for each dataset highlighted in bold. Following prior work [22], we report the 95% confidence intervals over five runs. In the table, the rows represent different models and the columns represent different datasets. As shown in Table 4, ProgNet achieves overall competitive accuracy. ProgNet achieves the best accuracy for all datasets except for BBBP. In the BBBP dataset, ProgNet achieves the second-best accuracy.

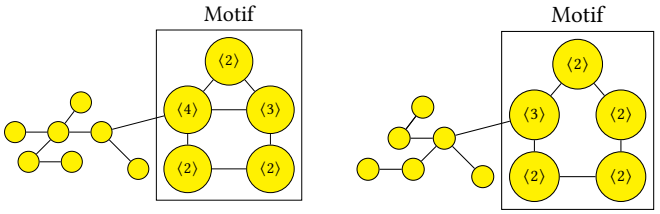
Table 5 compares the fidelity scores (lower is better) with 95% confidence intervals over five runs. As the fidelity score can vary depending on the chosen explanation size (e.g., user-defined threshold) except for PL4XGL, we measure the fidelity score matching the explanation size of PL4XGL. As the table shows, ProgNet achieves competitive *Fidelity* scores.

Table 4: Accuracy comparison on the eight graph classification datasets.

	NCI1	PTC	MUTAG	BA-2Motifs	PROTEINS	BBBP	BACE	Mutagenicity
GIN	79.5 ± 7.9	58.8 ± 7.3	93.0 ± 3.4	**100.0 ± 0.0**	66.1 ± 4.2	86.5 ± 0.1	81.4 ± 4.0	79.1 ± 0.1
GCN	78.3 ± 7.5	69.9 ± 7.3	84.0 ± 1.1	**100.0 ± 0.0**	71.0 ± 1.4	85.0 ± 0.8	78.1 ± 4.3	79.1 ± 3.8
GAT	64.3 ± 7.2	57.0 ± 7.7	75.0 ± 0.0	85.4 ± 7.5	68.3 ± 4.3	83.7 ± 0.8	64.6 ± 1.3	68.7 ± 1.6
PL4XGL	72.7 ± 0.0	67.6 ± 0.0	**100 ± 0.0**	**100 ± 0.0**	62.1 ± 0.0	86.3 ± 0.0	82.2 ± 0.0	76.9 ± 0.0
GSAT	72.0 ± 1.2	67.0 ± 2.1	87.0 ± 4.9	**100.0 ± 0.0**	**71.1 ± 0.7**	**87.2 ± 0.7**	78.1 ± 1.2	73.6 ± 1.5
ProgNet	**81.5 ± 0.8**	**72.3 ± 3.8**	**100.0 ± 0.0**	**100.0 ± 0.0**	**71.1 ± 2.7**	86.7 ± 1.7	**82.8 ± 1.4**	**79.7 ± 0.6**

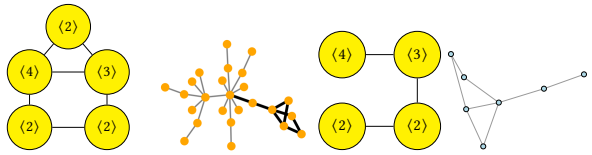
Table 5: Fidelity score comparison on the eight graph classification datasets.

Method	NCI1	PTC	MUTAG	BA-2Motifs	PROTEINS	BBBP	BACE	Mutagenicity
PL4XGL	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
GSAT	0.25 ± 0.05	0.16 ± 0.09	0.31 ± 0.03	0.52 ± 0.0	0.61 ± 0.11	0.16 ± 0.08	0.30 ± 0.03	0.25 ± 0.06
OrphicX	0.42 ± 0.04	0.21 ± 0.04	0.15 ± 0.05	0.48 ± 0.0	0.56 ± 0.0	0.16 ± 0.0	0.40 ± 0.02	0.39 ± 0.01
SubgraphX	0.17 ± 0.0	0.14 ± 0.01	0.5 ± 0.0	0.21 ± 0.01	0.67 ± 0.01	0.18 ± 0.0	0.48 ± 0.0	0.35 ± 0.01
ProgNet	0.06 ± 0.11	0.02 ± 0.01	0.01 ± 0.01	0.0 ± 0.0	0.4 ± 0.12	0.19 ± 0.03	0.20 ± 0.04	0.18 ± 0.03



(a) An example graph of label 1 **(b) An example graph of label 0**

Figure 9: Example graphs in BA-2Motifs dataset.



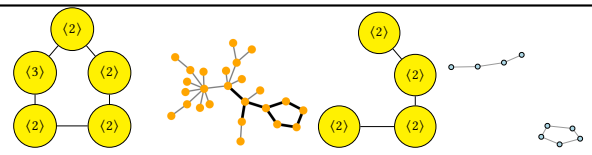
(a) ProgNet **(b) SubgraphX** **(c) PL4XGL** **(d) GSAT**

Figure 10: Explanations for graphs classified into label 1

B Explanations in the Synthetic Dataset

In this section, we present the subgraph explanations provided by ProgNet on the synthetic dataset BA-2Motifs, which is designed to evaluate explanation quality [11]. In this dataset, each node feature is its degree (e.g., a node with degree 3 has feature $\langle 3.0 \rangle$). The dataset has two labels (0 and 1), each determined by the presence of a specific motif. Figures 9a and 9b show example graphs. Label 1 corresponds to a house-shaped motif (Figure 9a) with five nodes: one roof node, two middle nodes (connected to each other), and two bottom nodes. One middle node connects to a randomly generated Barabási-Albert graph. Label 0 has a similar five-node structure, but the two middle nodes are not connected. The intuition behind this experiment is that if a model achieves high accuracy, it must have identified and used the key structures (i.e., the motifs) to make predictions [32]. Therefore, a correct explanation should also capture the motifs that the model has used for classification.

Figure 10 shows the subgraph explanations for a classification to label 1 generated by the four methods ProgNet, SubgraphX, PL4XGL, and GSAT. As the figure shows, all four methods successfully identify the motif. ProgNet (Figure 10a) exactly matches the five-node motif. SubgraphX (Figure 10b, bolded edges) captures the complete motif. PL4XGL (Figure 10c) identifies the key structural pattern of the motif of label 1. GSAT (Figure 10d) also correctly identifies the house-shaped motif of label 1.



(a) ProgNet **(b) SubgraphX** **(c) PL4XGL** **(d) GSAT**

Figure 11: Explanations for graphs classified into label 0

Figure 11 shows the subgraph explanations for a classification to label 0. Again, all four methods successfully identify the motif: ProgNet (Figure 11a) precisely captures the five-node structure, SubgraphX (Figure 11b) includes the complete motif, PL4XGL (Figure 11c) captures the structural pattern of the motif, and GSAT correctly identifies the motif of label 0.

C Cost Comparison

Table 6 presents the costs of the four techniques on the eight datasets (in minutes). The "Preprocessing" rows report the cost of the program-mining process. GNN+SubgraphX and GSAT do not include program mining, so their preprocessing cost is 0. The rows "Training", "Classification", and "Explanation" show the cost of training (including matching of programs to graphs and vocabulary selection), classification, and explanation, respectively. "Total" shows the sum of the training, classification, and explanation costs.

As shown in Table 6, the main bottleneck of ProgNet is the program-mining process. For example, ProgNet would cost much less if the program-mining process is excluded. We note that the program-mining cost can be reduced by about 80%, as discussed in Section 4.3, since ProgNet uses only a small portion of the mined programs. Mining only the necessary programs will make ProgNet significantly more scalable, and it is a promising direction for future work.

Table 6: Cost comparison among GNN+SubgraphX, PL4XGL, GSAT, and ProgNet in minutes.

		GNN+ SubgraphX	PL4XGL	GSAT	ProgNet			GNN+ SubgraphX	PL4XGL	GSAT	ProgNet
MUTAG	Preprocessing	0.0	4.1	0.0	4.1	BBBP	Preprocessing	0.0	8.9	0.0	8.9
	Training	0.2	0.1	0.8	0.1		Training	0.2	0.1	2.8	0.5
	Classification	0.1	0.1	0.1	0.1		Classification	0.1	0.8	0.1	0.1
	Explanation	60.1	0.0	0.1	0.1		Explanation	160.9	0.0	0.2	0.5
	Total	60.4	4.3	1.0	4.4		Total	161.2	9.8	3.1	10
Mutagenicity	Preprocessing	0.0	273.1	0.0	273.1	BACE	Preprocessing	0.0	14.3	0.0	14.3
	Training	0.2	0.1	6.2	1.5		Training	0.2	0.1	2.2	0.3
	Classification	0.1	27.2	0.2	0.1		Classification	0.1	3.9	0.2	0.1
	Explanation	604.5	0.0	0.3	4.8		Explanation	136.0	0.0	0.3	0.9
	Total	604.8	300.4	6.7	279.5		Total	136.3	18.3	2.7	15.6
PROTEINS	Preprocessing	0.0	74.1	0.0	74.1	NCI1	Preprocessing	0.0	13.5	0.0	13.5
	Training	0.2	0.1	1.9	0.2		Training	0.1	0.1	5.1	1.1
	Classification	0.1	0.2	0.1	0.1		Classification	0.1	42.4	0.2	0.1
	Explanation	418.8	0.0	0.1	0.1		Explanation	551.1	0.0	0.5	12.2
	Total	419.1	74.4	2.1	74.4		Total	551.3	58.0	5.8	26.9
PTC	Preprocessing	0.0	1.5	0.0	1.5	BA-2Motifs	Preprocessing	0.0	210.7	0.0	210.7
	Training	0.1	0.1	0.9	0.1		Training	0.2	0.1	1.9	1.1
	Classification	0.1	0.1	0.1	0.1		Classification	0.1	0.2	0.1	0.1
	Explanation	8.7	0.0	0.1	0.1		Explanation	93.1	0.0	0.2	0.1
	Total	8.9	1.7	1.1	1.8		Total	93.4	211.0	2.2	212.0

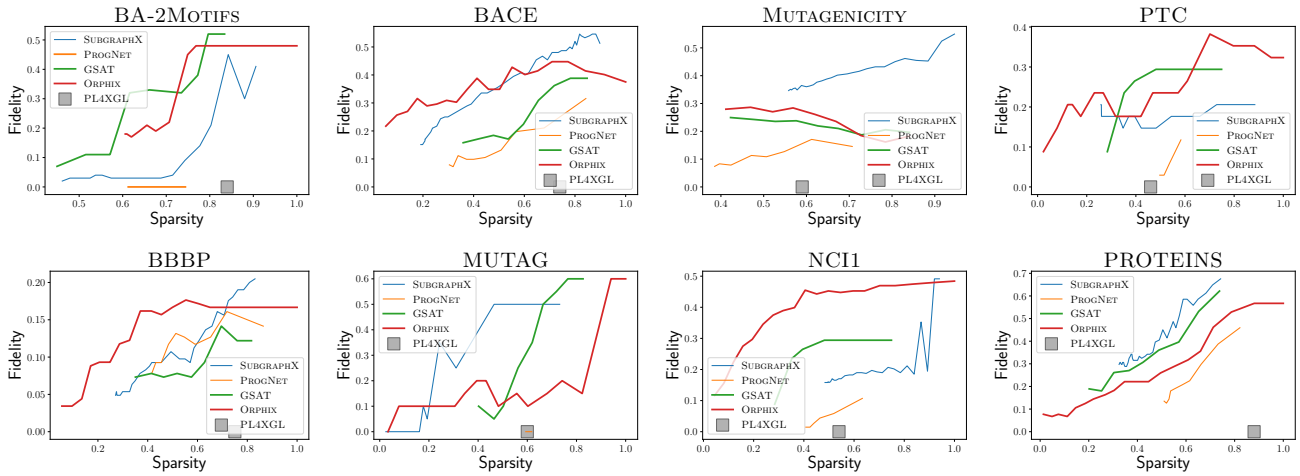


Figure 12: Sparsity-fidelity balance across the eight benchmarks. For the x-axis, higher values indicate simpler explanations. For the y-axis, lower values indicate more faithful explanations.

D Balance between Sparsity and Fidelity

Figure 12 shows how the balance between sparsity and fidelity changes on the eight datasets. In the figure, the x-axis represents the *Sparsity* measuring the simplicity of the explanations defined as follows:

$$Sparsity = \frac{1}{N}\sum_{i=1}^{N}(1 - \frac{|m_i|}{|M_i|}) \qquad \text{(higher is better)}$$

where N is the number of explained classifications, $|m_i|$ indicates the number of nodes in the explanation subgraph, and $|M_i|$ indicates the number of nodes in the original graph. In *Sparsity*, higher is better, implying greater simplicity.

As the figure shows, ProgNet achieves overall competitive balance between *Sparsity* and *Fidelity*. In the datasets BA-2Motifs, BACE, PTC, and NCI1, for example, ProgNet achieves overall the second-best fidelity after PL4XGL, which is inherently designed to achieve optimal *Fidelity*.