Harnessing the Power of Large Language Models for Inventive Graph Data Augmentation

Anonymous submission

Abstract

Graph-structured data is pervasive across various domains, but its complexity poses significant challenges for machine learning models. Effective graph data augmentation (GDA) is essential for improving model generalization, especially in scenarios with limited labeled data. However, traditional GDA methods are often predefined and lack the flexibility to adapt to the unique structure of different graph datasets. In this work, we introduce InventGDA, a novel framework that autonomously invents graph data augmentation strategies using pretrained large language models (LLMs). Unlike existing methods that rely on predefined augmentation techniques, InventGDA dynamically discovers and refines new strategies through a combination of prompt engineering and structured search operators. This approach ensures the generation of diverse and contextually relevant augmentations tailored to specific graph learning tasks. Extensive experiments on benchmark datasets demonstrate that InventGDA consistently outperforms state-of-the-art methods, offering superior generalization and robustness. Additionally, we provide indepth analyses and visualizations to better understand the discovered strategies, offering valuable insights into their effectiveness. InventGDA represents a significant advancement in the field of graph data augmentation, offering a flexible and powerful approach to autonomously inventing augmentation strategies.

Introduction

Graph-structured data is increasingly prevalent across diverse domains such as social networks, bioinformatics, computer vision, and natural language processing (Bronstein et al. 2017; Hamilton, Ying, and Leskovec 2017). The complexity and irregularity of graph data pose unique challenges for developing effective machine learning models. One key technique to address these challenges is *Graph Data Augmentation* (GDA), which enhances the generalization and robustness of graph-based models, particularly in scenarios with limited labeled data (Ding et al. 2018; Rong et al. 2020).

Traditional data augmentation methods, such as those used in image processing (e.g., rotation, flipping), are not directly applicable to graph data due to its discrete, relational nature. In graph domains, augmentations must preserve the inherent topological and structural properties while introducing meaningful variability to benefit the learning process (You et al. 2020a). This requirement has led to the development of various graph-specific augmentation strategies, including node dropping, edge perturbation, and subgraph sampling. However, designing effective graph augmentations remains a challenging task, as these strategies must be tailored to the specific characteristics of different datasets and learning objectives.

In recent years, the success of *Large Language Models* (LLMs) in natural language processing (NLP) has sparked interest in their potential applications across other domains (Brown et al. 2020; Devlin et al. 2018). LLMs have demonstrated remarkable capacities in tasks such as text generation, translation, and summarization. However, directly applying LLMs to graph data is not straightforward due to fundamental differences between language and graphs. These differences include:

1. **Structural Complexity:** Graphs encapsulate intricate relationships between entities (nodes) and their connections (edges), which are fundamentally different from the sequential nature of text. LLMs, which are primarily trained on linear sequences of tokens, struggle to directly interpret and manipulate the non-linear structures inherent in graph data (Kipf and Welling 2016a).
2. **Diverse Data Types:** Graphs often involve heterogeneous data, including node features, edge attributes, and global graph properties. LLMs are not inherently equipped to process such multi-modal and interconnected data types, making it challenging to generate meaningful graph augmentations (Veličković et al. 2018a).
3. **Lack of Graph-Specific Knowledge:** While LLMs are trained on vast amounts of textual data, they lack the domain-specific knowledge needed to understand and manipulate graph structures effectively (Hu et al. 2020). This limitation makes it difficult for LLMs to generate augmentation strategies that are both structurally valid and semantically meaningful within the context of graph learning.

Motivation and Contributions

Given these challenges, there is a clear need for a framework that can bridge the gap between the generative capabilities of LLMs and the structural complexities of graph data. This leads to our central motivation: *How can we leverage the power of LLMs to autonomously generate and refine graph data augmentation strategies, while overcoming the inherent challenges posed by graph structures?*

To address this, we propose **InventGDA**, a novel framework that combines the generative strengths of LLMs with a structured, iterative approach to graph data augmentation. InventGDA is designed to adaptively generate and refine graph augmentation strategies that are specifically tailored to the demands of graph learning tasks. The framework ensures that the augmentations are not only diverse but also contextually relevant, thereby maximizing the performance of *Graph Neural Networks* (GNNs) across various domains.

Our key contributions are as follows:

- **A Novel Integration Framework:** We introduce InventGDA, which effectively integrates LLMs with graph data augmentation techniques, addressing the critical challenge of generating graph-specific augmentations that enhance model performance.
- **Adaptive and Context-Aware Augmentation Generation:** We develop LLM-based search operators that dynamically generate, refine, and select augmentation strategies. These operators leverage prompt engineering to ensure that the generated strategies are aligned with the structural requirements of the graph data and the specific learning objectives.
- **Comprehensive Evaluation and Interpretability:** Extensive experiments on benchmark datasets show that InventGDA consistently outperforms state-of-the-art methods, while novel visualizations offer insights into how LLMs generate and refine effective graph augmentation strategies.

Related Work

Graph Data Augmentation

Graph data augmentation enhances self-supervised learning in graph-based models through strategic manipulation of graph structures and node features. We categorize these techniques into four primary classes:

- **Generation-based Methods:** These methods reconstruct graphs or node features to enhance data. Graph Autoencoders (GAE) (Kipf and Welling 2016b) exemplify this approach, learning node representations through graph structure reconstruction. However, the diversity of decoders and loss functions introduces significant complexity.
- **Auxiliary Property-based Methods:** Leveraging inherent graph properties like node degree or clustering coefficients, these methods synthesize new data. Graph property mining (You et al. 2020b) demonstrates this approach, although the heterogeneity of auxiliary attributes complicates method selection.
- **Contrast-based Methods:** Contrastive learning generates data by creating positive and negative sample pairs, maximizing consistency between local and global representations. Deep Graph Infomax (DGI) (Veličković et al. 2018b) stands as a prominent example in this category.
- **Hybrid Methods:** These approaches integrate strategies from multiple categories. GRACE (Zhu et al. 2020), for instance, combines node feature and graph structure perturbations to foster diverse data representations.

Traditional methods often rely on manually crafted heuristics or domain-specific rules, leading to high algorithmic complexity and design challenges. Our novel InventGDA approach leverages large language models to automate the generation of effective graph self-supervised learning algorithms, addressing these limitations.

Large Language Models

Large Language Models (LLMs) have emerged as powerful tools for automating complex algorithmic designs across various domains. Their applications span evolutionary algorithms (Yang et al. 2024; Liu et al. 2023; Guo et al. 2024), neural architecture search (Chen, Dohan, and So 2024; Nasir et al. 2024; Jawahar et al. 2023), and combinatorial optimization (Wang et al. 2024). LLMs have demonstrated proficiency in automating processes such as selection, crossover, and mutation in evolutionary algorithms, and enhancing techniques in neural architecture search and Monte Carlo Tree Search (Zhao, Lee, and Hsu 2024). Models like BERT (Devlin et al. 2018), GPT (Brown et al. 2020; Radford et al. 2018, 2019), and T5 (Raffel et al. 2020) have exhibited exceptional performance in tasks ranging from source code generation to neural network design. As well as EoH (Liu et al. 2024), it fully demonstrates the talent of large language models in heuristic learning tasks.

These advancements suggest LLMs could significantly enhance graph data augmentation by introducing more efficient and innovative strategies. Our novel InventGDA algorithm integrates LLMs' capabilities with heuristic learning, aiming to streamline augmentation, reduce manual strategy dependence, and improve the adaptability and effectiveness of graph-based learning models.

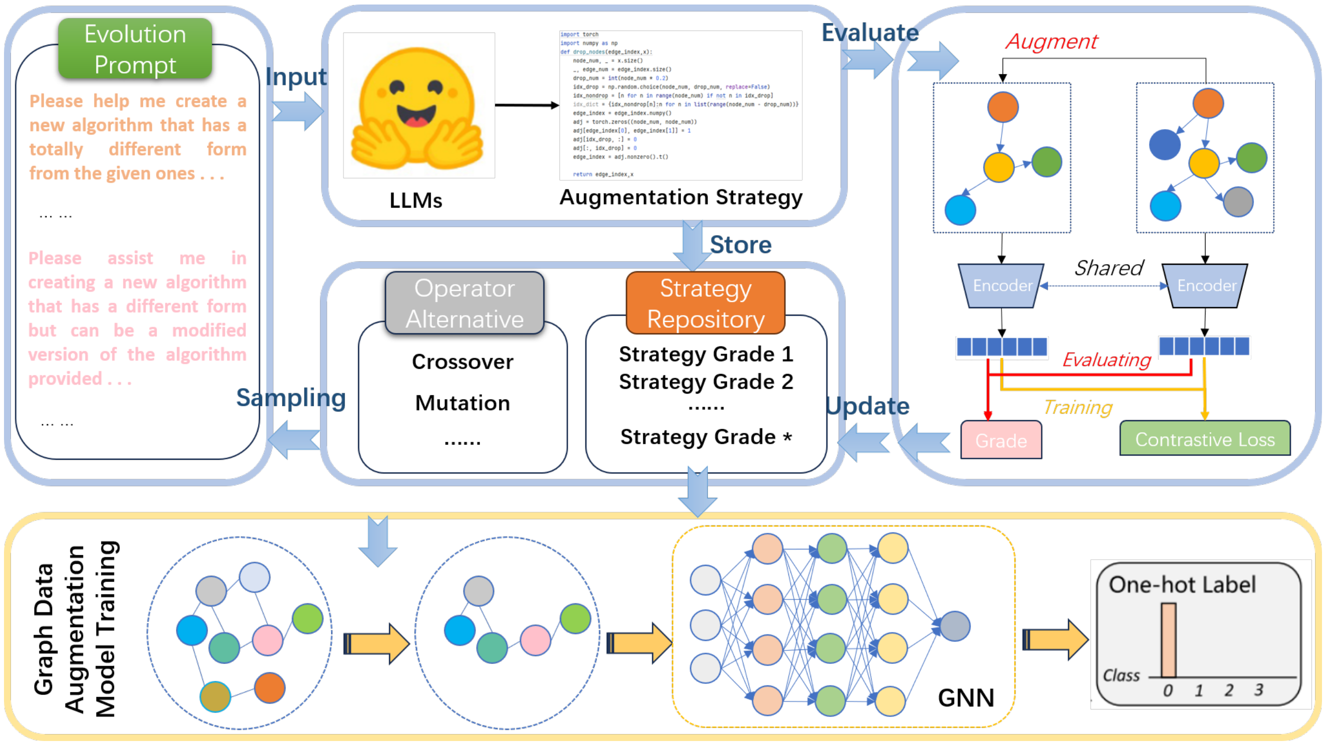
Here is the translation of the provided content into English, maintaining the more concise formalization and improving logical and contextual consistency:

Problem Definition and Preliminaries

Problem Definition: Graph Data Augmentation

Graph-structured data is represented by $G = (V, E)$, where V denotes the set of vertices and $E \subseteq V \times V$ represents the set of edges. Due to the inherent complexity of graph data, it presents unique challenges for machine learning models. The goal of Graph Data Augmentation (GDA) is to generate a set of augmented graphs $\{\tilde{G}_i\}$ from the original graph G, such that each augmented graph $\tilde{G}_i$ maintains semantic consistency with G while introducing sufficient variability to enhance model generalization and robustness, especially in scenarios with limited labeled data.

One of the primary challenges in GDA is the complexity and domain-specificity of graph data. Unlike image data, where augmentations such as rotation or flipping are intuitive and straightforward, graph data requires transformations that preserve relational and topological properties



Please help me create a new algorithm that has a totally different form from the given ones . . .

Please assist me in creating a new algorithm that has a different form but can be a modified version of the algorithm provided . . .

```
import torch
import numpy as np
def drop_nodes(edge_index,x):
    node_num, _ = x.size()
    _, edge_num = edge_index.size()
    drop_num = int(node_num * 0.2)
    idx_drop = np.random.choice(node_num, drop_num, replace=False)
    idx_nondrop = [n for n in range(node_num) if not n in idx_drop]
    idx_dict = {idx_nondrop[n]:n for n in list(range(node_num - drop_num))}
    edge_index = edge_index.numpy()
    adj = torch.zeros((node_num, node_num))
    adj[edge_index[0], edge_index[1]] = 1
    adj[idx_drop, :] = 0
    adj[:, idx_drop] = 0
    edge_index = adj.nonzero().t()

    return edge_index,x
```

Figure 1: Overview of InventGDA. InventGDA utilizes pretrained large language models to automatically search for optimal augmentation strategies tailored for graph data representation learning.

while inducing meaningful variations. These transformations must be carefully designed to ensure they contribute positively to model training, thereby enhancing the overall performance of graph-based models.

By addressing these challenges, effective GDA can significantly improve the generalization capabilities of graph neural networks and other graph-based learning algorithms.

Formalization of Graph Data Augmentation Techniques

To address the problem systematically, we define a set of augmentation operators $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_m\}$ that operate on the graph G to produce augmented versions $\tilde{G}_i = \mathcal{T}_i(G)$. These operators are integral to our proposed framework and encapsulate distinct aspects of graph variability:

Node Dropping ($\mathcal{T}_{ND}$) The operator $\mathcal{T}_{ND}$ is defined as:

$$\mathcal{T}_{ND}(G) = \tilde{G}_{ND} = (V', E'),$$

where $V' \subseteq V$, $E' = \{(u, v) \in E \mid u, v \in V'\}$. This operator stochastically removes a subset of vertices $V - V'$ along with their adjacent edges, under the hypothesis that the graph's global semantic structure is resilient to the loss of certain local components.

Edge Perturbation ($\mathcal{T}_{EP}$) The operator $\mathcal{T}_{EP}$ modifies the edge set E of the graph:

$$\mathcal{T}_{EP}(G) = \tilde{G}_{EP} = (V, E''),$$

where $E'' = E \cup E_{add} \setminus E_{remove}$, $E_{add} \subseteq (V \times V) \setminus E$, $E_{remove} \subseteq E$. This operator introduces perturbations in the connectivity of G, assuming that the graph's high-level semantics remain stable despite alterations in its edge structure.

Node Masking ($\mathcal{T}_{NM}$) The operator $\mathcal{T}_{NM}$ modifies the node feature matrix X of G:

$$\mathcal{T}_{NM}(G) = \tilde{G}_{NM} = (V, E, X'),$$

where $X' = \{x'_i \mid x'_i = \mathbf{0} \text{ if } i \in V_{mask} \text{ else } x'_i = x_i\}$, $V_{mask} \subseteq V$. This operator masks the features of a subset of nodes, promoting the model's ability to infer the missing information from the remaining graph structure.

Subgraph Sampling ($\mathcal{T}_{SS}$) The operator $\mathcal{T}_{SS}$ generates a subgraph $\tilde{G}_{SS}$ by sampling a subset of vertices:

$$\mathcal{T}_{SS}(G) = \tilde{G}_{SS} = (V_{sub}, E_{sub}),$$

where $V_{sub} \subseteq V$, $E_{sub} = \{(u, v) \in E \mid u, v \in V_{sub}\}$. It assumes that the intrinsic semantics of G are well captured by its local substructures, which can be effectively sampled through techniques like random walks.

Self-Supervised Learning in Graph Representation

In graph representation learning, the primary objective is to construct robust embeddings $\mathbf{z}_v$ for each vertex $v \in V$, or a global embedding $\mathbf{z}_G$ for the entire graph G. These embeddings aim to encapsulate the inherent structural and feature information within the graph. Under the self-supervised

Algorithm 1: InventGDA Algorithm for Graph Data Augmentation

Input: Graph $G = (V, E)$; Population size N_p; Generation limit T_{max}; Augmentation operators $\Omega = \{\omega_1, \ldots, \omega_m\}$; Operator probabilities $P(\Omega)$; Parent selection size k; Pre-trained LLM $\mathcal{L}$.
Output: Optimal GDA strategy S^*

```
 1: Initialize augmentation strategy population:
 2: for $i = 1, \ldots, N_p$ do
 3:     $S_i \leftarrow$ RandomSample($\Omega$, $P(\Omega)$)
 4:     $f(S_i) \leftarrow$ EvaluateStrategy($G$, $S_i$)
 5: end for
 6: $\mathcal{P} \leftarrow \{S_1, \ldots, S_{N_p}\}$
 7: for $t = 1, \ldots, T_{max}$ do
 8:     Parent Selection: $\mathcal{P}_{parent} \leftarrow$ SelectTopK($\mathcal{P}$, $k$)
 9:     Strategy Crossover: $S_{new} \leftarrow$ LLM-Crossover($\mathcal{L}$, $\mathcal{P}_{parent}$, $G$)
10:     for $\omega \in S_{new}$ do
11:         Operator Mutation: $\omega' \leftarrow$ LLM-Mutate($\mathcal{L}$, $\omega$, $G$)
12:         $S_{new} \leftarrow (S_{new} \setminus \{\omega\}) \cup \{\omega'\}$
13:     end for
14:     $f(S_{new}) \leftarrow$ EvaluateStrategy($G$, $S_{new}$)
15:     Population Update: $\mathcal{P} \leftarrow$ UpdatePopulation($\mathcal{P} \cup \{S_{new}\}$, $N_p$)
16: end for
17: return $S^* = argmax_{S \in \mathcal{P}} f(S)$
```

learning paradigm, the goal is to learn these embeddings by maximizing the similarity between augmented views of the same graph while minimizing the similarity between views of different graphs.

Given a set of augmented graphs $\{\tilde{G}_1, \tilde{G}_2, \ldots, \tilde{G}_k\}$ derived from G, the objective is to learn embeddings $\mathbf{z}_{\tilde{G}_i}$ that satisfy:

$$\mathcal{L}_{contrastive} = -\log \frac{\exp(\text{sim}(\mathbf{z}_{\tilde{G}_i}, \mathbf{z}_{\tilde{G}_j})/\tau)}{\sum_{k=1}^{2N} \mathbf{1}_{[k \neq i]} \exp(\text{sim}(\mathbf{z}_{\tilde{G}_i}, \mathbf{z}_{\tilde{G}_k})/\tau)} \tag{1}$$

where $\text{sim}(\cdot, \cdot)$ denotes a similarity function such as cosine similarity, and τ is a temperature hyperparameter. The contrastive loss $\mathcal{L}_{contrastive}$ is computed over all positive pairs $(\tilde{G}_i, \tilde{G}_j)$, enhancing the model's ability to distinguish between similar and dissimilar graph views.

Adaptive Graph Data Augmentation Framework

To dynamically adapt data augmentation strategies to the specific characteristics of graphs and learning tasks, we propose an adaptive GDA framework. This framework iteratively optimizes the augmentation operators $\mathcal{T}$ based on performance feedback from the learning process. The framework comprises the following key components:

Augmentation Strategy Initialization Let $\mathcal{P}_0 = \{\mathcal{T}_1^0, \mathcal{T}_2^0, \ldots, \mathcal{T}_m^0\}$ represent the initial population of augmentation strategies, where each strategy $\mathcal{T}_i^0$ is a combination of the basic augmentation operators $\mathcal{T}_{ND}, \mathcal{T}_{EP}, \mathcal{T}_{NM}, \mathcal{T}_{SS}$.

Iterative Strategy Optimization At each iteration t, the current population $\mathcal{P}_t$ is evaluated based on the performance of a graph neural network (GNN), with a fitness function $F : \mathcal{P}_t \rightarrow \mathbb{R}$ quantifying the quality of learned embeddings. New strategies $\mathcal{P}_{t+1}$ are generated through operations such as crossover and mutation, guided by the feedback from F.

Optimization and Convergence The iterative process continues until the augmentation strategies converge to an optimal set $\mathcal{P}^*$, formally defined as:

$$\mathcal{P}^* = \arg\max_{\mathcal{P}_t} F(\mathcal{P}_t) \tag{2}$$

$\mathcal{P}^*$ represents the set of strategies that maximize GNN performance across various downstream tasks. This optimization ensures that the augmentation process is adaptive and context-aware, leading to more effective graph representations.

InventGDA: A Heuristic Approach to Graph Data Augmentation

Structure

In this section, we introduce InventGDA, a heuristic algorithm that integrates large language models (LLMs) with graph data augmentation techniques. The whole framework of it is illustrated in Figure 1. The proposed framework comprises two core components: a module for generating augmentation strategies using LLMs and a module for training and evaluating graph learning models. Our approach is shown in algorithm 1.

Generating Augmentation Strategies with LLMs The first component of the framework focuses on the automated generation of graph data augmentation strategies, shown in Figure 1 (left, blue). Leveraging a pretrained large language model, we implement prompt engineering techniques to guide the LLM in producing a diverse set of strategies. These strategies are expressed in both natural language and executable code, allowing for seamless integration into the graph learning process. The generated strategies are collected in a pool, from which the most suitable ones are selected based on their effectiveness in enhancing the performance of graph neural networks.

Training and Evaluation with Dynamic Strategy Selection The second component of InventGDA involves the dynamic selection and application of data augmentation strategies during the training of a graph neural network (GNN), shown in Figure 1 (right, blue). At the beginning of each training epoch, the algorithm selects an augmentation strategy based on the current performance metrics, such as accuracy. This adaptive approach ensures that the selected strategies are aligned with the training objectives, thereby optimizing the GNN's performance over time. The interaction between the LLM-generated strategies and the GNN model forms a feedback loop that continually refines the augmentation process.

Algorithm 2: Graph Augmentation Strategy Evaluation

Input: Graph dataset $\mathcal{D} = \{G_1, \ldots, G_n\}$; Augmentation strategy S;
Contrastive learning model $\mathcal{M}_\theta$; Number of epochs E;
Classification model $\mathcal{C}$; Train-test split ratio α. **Output**: Fitness score $f(S)$.

```
1: $\mathcal{D}_{train}, \mathcal{D}_{test} \leftarrow$ SplitDataset($\mathcal{D}$, $\alpha$)
2: for $e = 1$ to $E$ do
3:     for $G_i \in \mathcal{D}_{train}$ do
4:         $\tilde{G}_i^1, \tilde{G}_i^2 \leftarrow S(G_i)$ {Apply augmentation strategy}
5:         $z_i^1, z_i^2 \leftarrow \mathcal{M}_\theta(\tilde{G}_i^1), \mathcal{M}_\theta(\tilde{G}_i^2)$ {Generate embeddings}
6:         $\mathcal{L}_{contrast} \leftarrow$ ContrastiveLoss($z_i^1$, $z_i^2$)
7:         UpdateModel($\mathcal{M}_\theta$, $\mathcal{L}_{contrast}$)
8:     end for
9: end for
10: $\mathcal{Z}_{train} \leftarrow \{\mathcal{M}_\theta(G_i) | G_i \in \mathcal{D}_{train}\}$
11: $\mathcal{Z}_{test} \leftarrow \{\mathcal{M}_\theta(G_i) | G_i \in \mathcal{D}_{test}\}$
12: TrainClassifier($\mathcal{C}$, $\mathcal{Z}_{train}$)
13: $f(S) \leftarrow$ Accuracy($\mathcal{C}$, $\mathcal{Z}_{test}$)
14: return $f(S)$
```

Assessing Strategy Performance

The strategy evaluation module of the algorithm is also the fitness calculation module as shown in the algorithm 2. To evaluate the performance of candidate data augmentation strategies, they are integrated into the model training process. A small portion of the training set undergoes additional training, and the validation set's accuracy is used to measure the algorithm's fitness score. Suppose the data augmentation function generated by the LLM is denoted as $\mathcal{F}_{\text{aug}}$ and the training-testing function as $\mathcal{T}$. The evaluation process can be represented as:

$$\mathcal{F}_{\text{itness}}(\mathcal{F}_{\text{aug}}) = \mathcal{T}_{ND}(\mathcal{F}_{\text{aug}}, \mathcal{E}_{\text{ckp}}, \mathcal{N}_{\text{ft}})$$

where $\mathcal{E}_{\text{ckp}}$ is the starting checkpoint epoch number and $\mathcal{N}_{\text{ft}}$ is the number of finetuning epochs. The function $\mathcal{T}$ injects $\mathcal{F}_{\text{aug}}$ into the training flow:

$$\mathcal{T}_{ND}(\mathcal{F}_{\text{aug}}, \mathcal{E}_{\text{ckp}}, \mathcal{N}_{\text{ft}}) = \mathcal{A}_{\text{cc}_{\text{val}}}(\mathcal{F}_{\text{ine-tune}}(\mathcal{F}_{\text{aug}}, \mathcal{D}_{\text{train}}, \mathcal{E}_{\text{ckp}}, \mathcal{N}_{\text{ft}}))$$

The Finetune function starts training from the $\mathcal{E}_{\text{ckp}}$ checkpoint and performs $\mathcal{N}_{\text{ft}}$ epochs of training on the training set $\mathcal{D}_{\text{train}}$. At the beginning of each epoch, it dynamically selects data augmentation methods for each class using $\mathcal{F}_{\text{aug}}$:

$$\mathcal{A}_c^{(t)}, \mathcal{E}_c^{(t)} = \mathcal{F}_{\text{aug}}(\mathcal{W}_c^{(t-1)}, \mathcal{A}_c^{(t-1)}, \mathcal{H}_c^{(t-1)}, \mathcal{A}_c^{(t-1)}, \mathcal{E}_c^{(t-1)}, t)$$

Here, $\mathcal{A}_c(t)$ represents the DA selection matrix for class c at time t, $\mathcal{E}_c(t)$ represents the corresponding augmentation intensity, $\mathcal{W}_c(t-1)$ represents the weights of each augmentation method on class c from the previous time step, $\mathcal{A}_c(t-1)$ represents the accuracy of class c at time $t-1$, and $\mathcal{H}_c(t-1)$ represents the historical accuracy of class c when using different augmentation methods in the previous step. After training, the model is evaluated using the non-augmented validation set $\mathcal{D}_{\text{val}}$, and the overall accuracy Acc_{val} is obtained as the fitness score for $\mathcal{F}_{\text{aug}}$:

$$\mathcal{F}_{\text{itness}}(\mathcal{F}_{\text{aug}}) = \mathcal{A}_{\text{cc}_{\text{val}}}(\mathcal{F}_{\text{inetune}}(\mathcal{F}_{\text{aug}}, \mathcal{D}_{\text{train}}, \mathcal{E}_{\text{ckp}}, \mathcal{N}_{\text{ft}}), \mathcal{D}_{\text{val}})$$

A higher fitness score indicates better performance of the algorithm on graph neural network distributions. By injecting candidate algorithms into the real training process and evaluating them on the validation set, we can accurately and efficiently measure their actual ability to address graph neural network downstream problems.

LLM-based Search Operators

The framework utilizes pretrained language models (PLMs) to autonomously generate data augmentation algorithms. To align the PLM's outputs with specific requirements, we employ prompt engineering techniques and carefully design a series of prompts. These prompts encompass task descriptions, input-output formats, novelty requirements, and other relevant prior knowledge, constraining the PLM's generation process within the desired search space.

We have designed the following three types of search operators, each corresponding to distinct prompt templates:

- **Initialization Operator $\mathcal{I}_{\mathcal{O}}$:**
  Based on the task description prompt $\mathcal{P}_{\text{task}}$ and the data augmentation knowledge base (K), a set of randomly initialized population algorithms ($\{\mathcal{A}_i^{(0)}\}_{i=1}^{\mathcal{N}}$) is generated.
- **Crossover Operator $\mathcal{E}_{\mathcal{O}}$:**
  Building on $\mathcal{P}_{\text{task}}$, $\mathcal{N}_p$ parent algorithms ($\{\mathcal{A}_i^{(t)}\}_{i=1}^{\mathcal{N}_p}$) from the current population are used as references, along with the incorporation of knowledge base ($\mathcal{K}$). The PLM generates $\mathcal{N}_e$ new algorithms ($\{\mathcal{A}_j^{(t)}\}_{j=1}^{\mathcal{N}_e}$) that are distinct in both form and logic from existing algorithms, thereby expanding the search space.
- **Mutation Operator $\mathcal{M}_{\mathcal{O}}$:**
  Based on $\mathcal{P}_{\text{task}}, \mathcal{N}_m$ are selected from the current population, and local improvement directions are provided. The PLM generates a mutated algorithm ($\mathcal{A}_i^{(t)}$) for each $\mathcal{A}_i^{(t)}$ within its neighborhood for further exploration.

For example, the prompt $\mathcal{P}_E$ for the crossover operator can be expressed as follows:

$$\begin{aligned}\mathcal{P}_{\mathcal{E}}(\mathcal{P}_{\text{task}}, \{\mathcal{A}_i^{(t)}\}_{i=1}^{\mathcal{N}_p}, \mathcal{K}, \mathcal{D}_{\text{func}}) = {} & \mathcal{P}_{\text{task}} + \mathcal{P}_{\text{ref}}(\{\mathcal{A}_i^{(t)}\}_{i=1}^{\mathcal{N}_p}) \\ & + \mathcal{P}_{\text{know}}(\mathcal{K}) + \mathcal{P}_{\text{diff}} \\ & + \mathcal{P}_{\text{format}}(\mathcal{D}_{\text{func}}),\end{aligned}$$

where $\mathcal{P}_{\text{task}}$ represents the task description, $\{\mathcal{A}_i^{(t)}\}_{i=1}^{\mathcal{N}_p}$ represents the $\mathcal{N}_p$ parent algorithms, and $\mathcal{D}_{\text{func}}$ represents the domain of the objective function. $\mathcal{P}_{\text{ref}}$ formats the parent algorithms into reference code, $\mathcal{P}_{\text{diff}}$ requires the generation of new algorithms that differ completely from the existing ones, and $\mathcal{P}_{\text{format}}$ specifies the input and output of the objective function.

Once we have the prompt $\mathcal{P}_{\mathcal{E}}$, it is input into the pretrained language model, which generates $\mathcal{N}_e$ new crossover algorithms:

$$\{\mathcal{A}_j^{(t)}\}_{j=1}^{\mathcal{N}_e} = \mathcal{L}(\mathcal{P}_{\mathcal{E}}(\mathcal{P}_{\text{task}}, \{\mathcal{A}_i^{(t)}\}_{i=1}^{\mathcal{N}_p}, \mathcal{K}, \mathcal{D}_{\text{func}})).$$

Each $\mathcal{A}_j^{(t)}$ typically consists of a natural language description of the algorithm and its corresponding Python code implementation. Similarly, the prompts $\mathcal{P}_{\mathcal{I}}$ and $\mathcal{P}_{\mathcal{M}}$ for the initialization operator (I) and the mutation operator (M) can

Methods	NCI1	PROTEINS	DD	MUTAG	COLLAB	RDT-B	RDT-M5K	IMDB-B	Mean-Rank
GL	-	-	-	81.66±2.11	-	77.34±0.18	41.01±0.17	65.87±0.98	9.0_{10}
WL	80.01±0.50	72.92±0.56	-	80.72±3.00	-	68.82±0.41	46.06±0.21	72.30±3.44	6.7_{8}
DGK	**80.31**±0.46	73.30±0.82	-	87.44±2.72	-	78.04±0.39	41.27±0.18	66.96±0.56	6.0_{6}
node2vec	54.89±1.61	57.49±3.57	-	72.63±10.20	-	-	-	-	9.0_{10}
sub2vec	52.84±1.47	53.03±5.55	-	61.05±15.80	-	71.48±0.41	36.68±0.42	55.26±1.54	11.2_{11}
graph2vec	73.22±1.81	73.30±2.05	-	83.15±9.25	-	75.78±1.03	47.86±0.26	71.10±0.54	6.7_{8}
MVGRL	-	-	-	75.40±7.80	-	82.00±1.10	-	63.60±4.20	8.7_{9}
InfoGraph	76.20±1.06	74.44±0.31	72.85±1.78	**89.01**±1.13	70.65±1.13	82.50±1.42	53.46±1.03	**73.03**±0.87	3.1_{2}
JOAO	78.07±0.47	74.55±0.41	77.32±0.54	87.35±1.02	69.50±0.36	85.29±1.35	55.74±0.63	70.21±3.08	4.0_{4}
JOAOv2	78.36±0.53	74.07±1.10	77.40±1.15	87.67±0.79	69.33±0.34	86.42±1.45	**56.03**±0.27	70.83±0.25	3.5_{3}
DualGraph	-	70.10±1.20	69.80±0.80	-	67.20±0.60	75.40±1.40	42.90±1.40	72.10±0.70	7.8_{9}
TGNN	-	71.00±0.70	70.80±0.90	-	67.70±0.40	76.30±1.30	43.80±1.00	72.80±1.70	6.2_{7}
GraphSpa	-	71.20±0.70	71.4±0.80	-	-	76.50±0.40	44.00±0.60	72.30±1.10	5.8_{5}
InventGDA-Seq	79.18±1.02	**75.20**±0.27	**78.41**±0.61	88.62±0.92	**71.29**±0.89	**88.12**±1.66	55.49±0.28	71.17±0.68	2.0_{1}

Table 1: Transfer learning results on TUDataset. Red numbers represent the top-3 accuracy (%), with **bold** indicating first place and underlined indicating second place. The compared results are from the published papers, and - indicates that results are not available in published papers. We select three well-performed foundational methods generated by our proposed InventGDA to form a sequence, referred to as InventGDA-Seq, from which algorithms can be dynamically chosen during the training process.

Dataset	None	Drop Nodes	Pert. Edges	Sub-graph	Mask Nodes	InventGDA-Basic
NCI1	73.11	78.98	78.52	78.02	76.44	**79.18**
DD	75.47	**77.96**	76.73	76.81	76.58	77.67
MUTAG	86.73	86.01	86.01	85.18	86.00	**86.89**
COLLAB	62.92	71.22	**71.52**	70.84	70.91	71.35
RDT-B	70.80	89.27	89.56	88.01	88.29	**90.37**
RDT-M5K	42.51	54.93	54.85	44.99	53.89	**55.57**
IMDB-B	66.50	**71.57**	71.33	71.22	71.40	71.33

Table 2: Comparison with other basic data augmentation techniques on TUdataset. Here, InventGDA-Basic means the best algorithm that generated by our proposed InventGDA.

be constructed in a similar manner, with the main difference being the introduced prior information.

Experiment

We evaluate the performance of our proposed InventGDA method on graph classification tasks using publicly available datasets. See the Appendix for more details and reproducibility.

Experimental Setup

Datasets We use eight diverse datasets from the TUDataset benchmark (Morris et al. 2020), including NCI1 (Wale, Watson, and Karypis 2008), PROTEINS (Borgwardt et al. 2005), DD (Dobson and Doig 2003), MUTAG (Debnath et al. 1991), COLLAB (Yanardag and Vishwanathan 2015), RDT-B (Yanardag and Vishwanathan 2015), RDT-M5K (Yanardag and Vishwanathan 2015), and IMDB-B (Yanardag and Vishwanathan 2015), covering various domains like bioinformatics and social networks.

Baselines We compare our method against several baselines: graph kernel methods (GL (Shervashidze et al. 2009), WL (Shervashidze et al. 2011), DGK (Yanardag and Vishwanathan 2015)), embedding techniques (node2vec (Grover and Leskovec 2016), sub2vec (Adhikari et al. 2018), graph2vec (Narayanan et al. 2017)), and GNN-based methods (MVGRL (Hassani and Khasahmadi 2020), InfoGraph (Sun et al. 2019), GraphCL (You et al. 2020a), JOAO (You et al. 2021), JOAOv2 (You et al. 2021), DualGraph (Luo et al. 2022), TGNN (Ju et al. 2023), GraphSpa (Ju et al. 2024)).

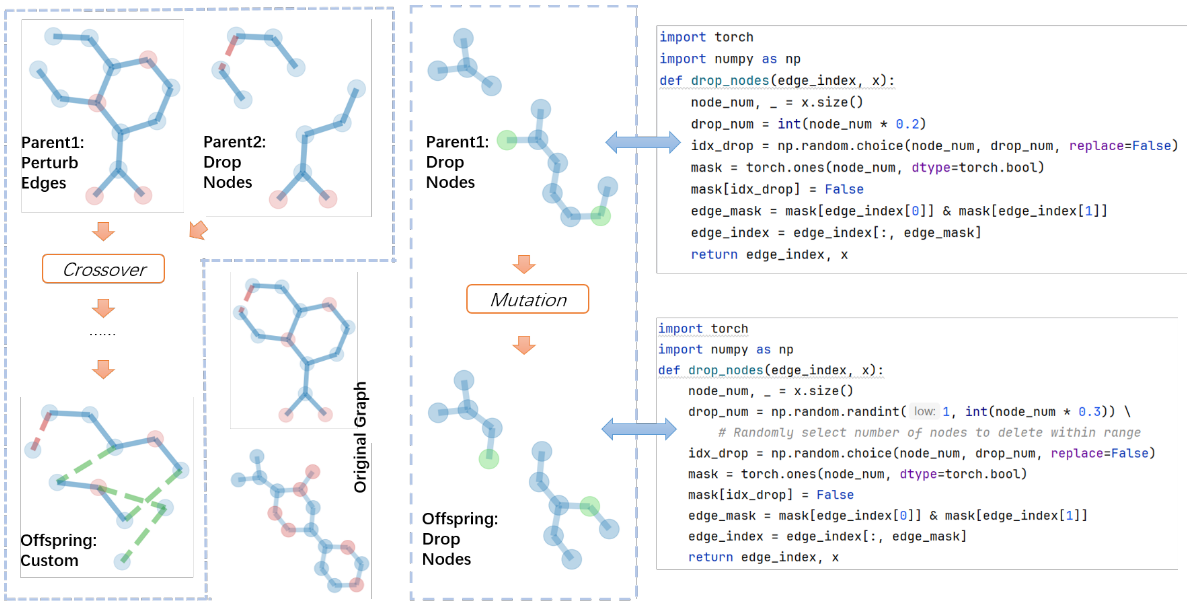
Implementation Details Experiments are implemented using PyTorch (Paszke et al. 2017) and conducted on an NVIDIA GeForce RTX 3090 GPU. We focus LLM-augmentation evolution experiments on the PROTEINS dataset, applying our method across all datasets.

- **GNN Structure and Self-supervised Learning**: We use a GIN (Xu et al. 2018) with three layers and 32 hidden dimensions, Adam optimizer with a learning rate of 0.1, and a batch size of 128.
- **LLM-based Strategy Generation**: GPT-3.5-turbo generates strategies. We perform one epoch of contrastive learning with LLM-generated augmentations to create positive and negative samples.
- **Transfer Learning**: An SVM classifier maps graph feature representations to labels, with dynamic selection of LLM-generated alternatives based on dataset distributions.

Experimental Results

As shown in Table 1, our approach consistently achieves top-2 accuracy across multiple datasets, outperforming fixed, human-designed augmentations (Table 2). This demonstrates InventGDA's effectiveness in generating high-quality augmentations.

Figure 2 illustrates the LLM evolution processes. The evolution leads to more complex augmentations, enhancing graph diversity and promoting self-supervised learning. Code snippets highlight subtle evolutionary changes.



```
import torch
import numpy as np
def drop_nodes(edge_index, x):
    node_num, _ = x.size()
    drop_num = int(node_num * 0.2)
    idx_drop = np.random.choice(node_num, drop_num, replace=False)
    mask = torch.ones(node_num, dtype=torch.bool)
    mask[idx_drop] = False
    edge_mask = mask[edge_index[0]] & mask[edge_index[1]]
    edge_index = edge_index[:, edge_mask]
    return edge_index, x
```

```
import torch
import numpy as np
def drop_nodes(edge_index, x):
    node_num, _ = x.size()
    drop_num = np.random.randint( low: 1, int(node_num * 0.3)) \
        # Randomly select number of nodes to delete within range
    idx_drop = np.random.choice(node_num, drop_num, replace=False)
    mask = torch.ones(node_num, dtype=torch.bool)
    mask[idx_drop] = False
    edge_mask = mask[edge_index[0]] & mask[edge_index[1]]
    edge_index = edge_index[:, edge_mask]
    return edge_index, x
```

Figure 2: Illustration of two evolution processes in the generation of basic data augmentation algorithms. Key changes in nodes and edges within the graph after operations are highlighted in different colors.

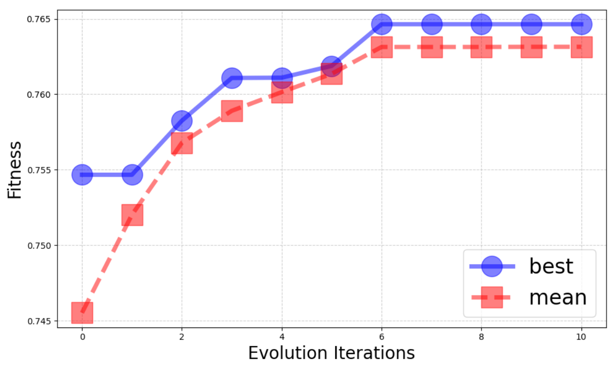


Figure 3: Evolution curve over 10 iterations on the PROTEINS dataset. Population size: 3. We report best and mean accuracies on graph classification tasks after each evolution iteration.

Analysis and Discussion

We conduct ablation experiments to validate the effectiveness of key components.

Can LLMs Interpret Graph Data Augmentation? Comparing the "None" setting with others in Figure 4, we observe that using LLM-generated data algorithms usually leads to higher accuracies for the self-supervised model on downstream tasks. This demonstrates that, even though large language models are not specifically designed for processing graph data, they can grasp fundamental principles and related operations of graph data augmentation through language descriptions.

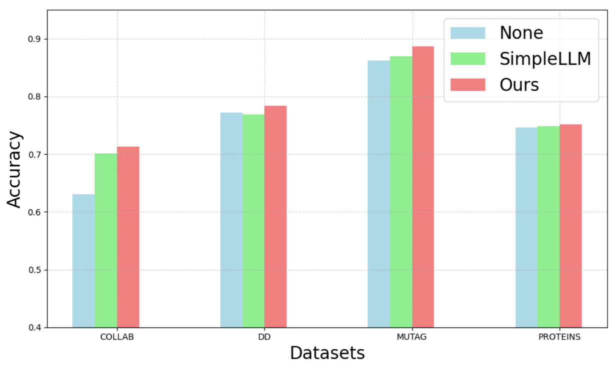


Figure 4: Ablation results on four datasets. "None": no augmentation, "SimpleLLM": LLM outputs without evolution.

Can Evolution Enhance Data Augmentation? Comparing "SimpleLLM" with "Ours" in Figure 4 shows that our evolution-based mechanism outperforms non-evolved LLM outputs. The "None" vs. "SimpleLLM" comparison on the DD dataset suggests that non-evolved augmentations may negatively impact learning. Figure 3 shows that fitness improves with evolution, indicating more effective data augmentation selection.

Conclusion

We propose a novel closed-loop system for graph learning. Our approach utilizes the powerful interpretation and generation ability of large language models. Extensive experiments have been conducted to validate our approach in improving the performance of self-supervised models on graph classification tasks.

References

Adhikari, B.; Zhang, Y.; Ramakrishnan, N.; and Prakash, B. A. 2018. Sub2vec: Feature learning for subgraphs. In *Advances in Knowledge Discovery and Data Mining: 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part II 22*, 170–182. Springer.

Borgwardt, K. M.; Ong, C. S.; Schönauer, S.; Vishwanathan, S.; Smola, A. J.; and Kriegel, H.-P. 2005. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1): i47–i56.

Bronstein, M. M.; Bruna, J.; LeCun, Y.; Szlam, A.; and Vandergheynst, P. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4): 18–42.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33: 1877–1901.

Chen, A.; Dohan, D.; and So, D. 2024. EvoPrompting: language models for code-level neural architecture search. *Advances in Neural Information Processing Systems*, 36.

Debnath, A. K.; Lopez de Compadre, R. L.; Debnath, G.; Shusterman, A. J.; and Hansch, C. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2): 786–797.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ding, K.; Huang, L.; Wei, Z.; and Wang, Y. 2018. Graph data augmentation for graph machine learning: A survey. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Dobson, P. D.; and Doig, A. J. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4): 771–783.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 855–864.

Guo, P.-F.; Chen, Y.-H.; Tsai, Y.-D.; and Lin, S.-D. 2024. Towards Optimizing with Large Language Models. arXiv:2310.05204.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

Hassani, K.; and Khasahmadi, A. H. 2020. Contrastive multi-view representation learning on graphs. In *International conference on machine learning*, 4116–4126. PMLR.

Hu, W.; Liu, B.; Gomes, J.; Zitnik, M.; Liang, P.; Pande, V.; and Leskovec, J. 2020. GPT-GNN: Generative Pre-Training of Graph Neural Networks. *arXiv preprint arXiv:2007.09296*.

Jawahar, G.; Abdul-Mageed, M.; Lakshmanan, L. V.; and Ding, D. 2023. LLM Performance Predictors are good initializers for Architecture Search. *arXiv preprint arXiv:2310.16712*.

Ju, W.; Luo, X.; Qu, M.; Wang, Y.; Chen, C.; Deng, M.; Hua, X.-S.; and Zhang, M. 2023. TGNN: A joint semi-supervised framework for graph-level classification. *arXiv preprint arXiv:2304.11688*.

Ju, W.; Mao, Z.; Qiao, Z.; Qin, Y.; Yi, S.; Xiao, Z.; Luo, X.; Fu, Y.; and Zhang, M. 2024. Focus on informative graphs! Semi-supervised active learning for graph-level classification. *Pattern Recognition*, 153: 110567.

Kipf, T. N.; and Welling, M. 2016a. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kipf, T. N.; and Welling, M. 2016b. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*.

Liu, F.; Lin, X.; Wang, Z.; Yao, S.; Tong, X.; Yuan, M.; and Zhang, Q. 2023. Large language model for multi-objective evolutionary optimization. *arXiv preprint arXiv:2310.12541*.

Liu, F.; Xialiang, T.; Yuan, M.; Lin, X.; Luo, F.; Wang, Z.; Lu, Z.; and Zhang, Q. 2024. Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model. In *Forty-first International Conference on Machine Learning*.

Luo, X.; Ju, W.; Qu, M.; Chen, C.; Deng, M.; Hua, X.-S.; and Zhang, M. 2022. Dualgraph: Improving semi-supervised graph classification via dual contrastive learning. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 699–712. IEEE.

Morris, C.; Kriege, N. M.; Bause, F.; Kersting, K.; Mutzel, P.; and Neumann, M. 2020. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*.

Narayanan, A.; Chandramohan, M.; Venkatesan, R.; Chen, L.; Liu, Y.; and Jaiswal, S. 2017. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*.

Nasir, M. U.; Earle, S.; Togelius, J.; James, S.; and Cleghorn, C. 2024. LLMatic: Neural Architecture Search via Large Language Models and Quality Diversity Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1110–1118.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch.

Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I.; et al. 2018. Improving language understanding by generative pre-training.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140): 1–67.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2020. Self-supervised graph representation learning via graph data augmentation. *arXiv preprint arXiv:2006.08273*.

Shervashidze, N.; Schweitzer, P.; Van Leeuwen, E. J.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9).

Shervashidze, N.; Vishwanathan, S.; Petri, T.; Mehlhorn, K.; and Borgwardt, K. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*, 488–495. PMLR.

Sun, F.-Y.; Hoffmann, J.; Verma, V.; and Tang, J. 2019. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018a. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Veličković, P.; Fedus, W.; Hamilton, W. L.; Liò, P.; Bengio, Y.; and Hjelm, R. D. 2018b. Deep graph infomax. *arXiv preprint arXiv:1809.10341*.

Wale, N.; Watson, I. A.; and Karypis, G. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14: 347–375.

Wang, H.; Feng, S.; He, T.; Tan, Z.; Han, X.; and Tsvetkov, Y. 2024. Can language models solve graph problems in natural language? *Advances in Neural Information Processing Systems*, 36.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Yanardag, P.; and Vishwanathan, S. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 1365–1374.

Yang, C.; Wang, X.; Lu, Y.; Liu, H.; Le, Q. V.; Zhou, D.; and Chen, X. 2024. Large Language Models as Optimizers. arXiv:2309.03409.

You, Y.; Chen, T.; Shen, Y.; and Wang, Z. 2021. Graph contrastive learning automated. In *International Conference on Machine Learning*, 12121–12132. PMLR.

You, Y.; Chen, T.; Sui, Y.; Zhao, T.; Shen, X.; and Wang, Z. 2020a. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33: 5812–5823.

You, Y.; Chen, T.; Wang, Z.; and Shen, Y. 2020b. When does self-supervision help graph convolutional networks? In *international conference on machine learning*, 10871–10880. PMLR.

Zhao, Z.; Lee, W. S.; and Hsu, D. 2024. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems*, 36.

Zhu, Y.; Xu, Y.; Yu, F.; Liu, Q.; Wu, S.; and Wang, L. 2020. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*.

Reproducibility Checklist

This paper:

- Includes a conceptual outline and/or pseudocode description of AI methods introduced (yes)
- Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results (yes)
- Provides well marked pedagogical references for less-familiare readers to gain background necessary to replicate the paper (yes)

Does this paper make theoretical contributions? (no)
If yes, please complete the list below.

- All assumptions and restrictions are stated clearly and formally. (yes/partial/no)
- All novel claims are stated formally (e.g., in theorem statements). (yes/partial/no)
- Proofs of all novel claims are included. (yes/partial/no)
- Proof sketches or intuitions are given for complex and/or novel results. (yes/partial/no)
- Appropriate citations to theoretical tools used are given. (yes/partial/no)
- All theoretical claims are demonstrated empirically to hold. (yes/partial/no/NA)
- All experimental code used to eliminate or disprove claims is included. (yes/no/NA)

Does this paper rely on one or more datasets? (yes)
If yes, please complete the list below.

- A motivation is given for why the experiments are conducted on the selected datasets (yes)
- All novel datasets introduced in this paper are included in a data appendix. (NA)
- All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (NA)
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations. (yes)
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available. (yes)
- All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientifically satisficing. (NA)

Does this paper include computational experiments? (yes)
If yes, please complete the list below.

- Any code required for pre-processing data is included in the appendix. (yes).
- All source code required for conducting and analyzing the experiments is included in a code appendix. (yes)
- All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (yes)
- All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (yes)
- If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results. (NA)
- This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks. (yes)
- This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. (yes)
- This paper states the number of algorithm runs used to compute each reported result. (yes)
- Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information. (yes)
- The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank). (yes)
- This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments. (NA)
- This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting. (NA)