Curriculum-GraphLLM: Joint Optimization of Architectures, Structures and Texts for Denoised Graph Neural Architecture Search

Xin Wang, *Member, IEEE,* Haibo Chen, Linxin Xiao, Zeyang Zhang and Wenwu Zhu, *Fellow, IEEE,*

Abstract—Discovering optimal graph neural network (GNN) architectures for various tasks is both labor-intensive and time-consuming. To reduce human effort, graph neural architecture search (GNAS) has recently been utilized to automatically identify effective GNN architectures for specific tasks, achieving competitive or even superior performance compared to manually designed architectures. However, existing GNAS methods fail to identify optimal architectures in the presence of *structural and semantic noise*, where structural noise refers to missing or redundant edges within the graph structure, and semantic noise denotes inaccurate node representations derived from ambiguous node features such as textual vagueness or semantic ambiguity. In this paper, we address this problem for the first time via theoretical analyses and empirical evaluations. We discover that existing differentiable GNAS methods typically select architectures based on task-relevant information hidden in the graph, being highly sensitive to *structural and semantic noise*, which results in suboptimal selection of GNN architectures under noise. To handle the *structural and semantic noise*, we propose Curriculum-GraphLLM, a novel graphLLM framework for joint optimization of architectures, structures, and texts for denoised graph neural architecture search. The core idea is to jointly optimize GNN architectures, graph structures, and textual semantics as a unified denoising process during architecture search. Specifically, we first develop a dynamic topology updating mechanism to adaptively adjust the graph structure. Then, we jointly optimize the GNN architecture and graph structure through a curriculum-based iterative updating approach. To further deal with semantic noise on text-attributed graphs (TAGs), we introduce LLMs as an auxiliary text modeling module to refine textual semantics and guide the co-optimization of text representations, graph structures, and GNN architectures. We conduct extensive experiments to show that our proposed Curriculum-GraphLLM achieves consistently competitive or superior performance compared with existing baselines, especially under structural and semantic noise.

Index Terms—Graph Neural Network, Neural Architecture Search, Graph Structure Learning, Curriculum Learning, GraphLLM.

1 Introduction

Graphs are commonly used to represent various types of relational data in real-world applications, such as social networks [1], e-commerce [2], traffic systems [3], and biochemistry [4]. To learn and extract knowledge from such graph-structured data, graph neural networks (GNNs) have been developed, including representative models such as GCN [5], GAT [6], and GIN [7]. GNNs utilize both the graph structure and node features by following a recursive message-passing scheme, where nodes aggregate information from their neighbors in each layer. This approach has achieved great success in various graph-related tasks [8]. However, identifying the most suitable GNN architecture for a specific task requires significant human effort and time. To address this issue, graph neural architecture search (GNAS) [9], [10], [11], [12] has been proposed, leveraging the principles of neural architecture search (NAS) [13], [14], [15] to automatically design optimal GNN architectures. This approach not only significantly reduces the need for human intervention but also enables the automatically generated GNNs to achieve performance that is competitive with, or even surpasses, manually designed GNNs across various tasks [16].

However, existing GNAS methods fail to identify optimal architectures in the presence of *structural and semantic noise*. Structural noise refers to missing or redundant edges within the graph structure, and semantic noise denotes inaccuracies in node embeddings derived from ambiguous node features such as textual vagueness or semantic ambiguity. Such noises commonly occur in real-world scenarios. For instance, social network graphs often contain missing edges due to incomplete information about user relationships, while redundant edges may result from noisy or inaccurate data collection. A typical instance of semantic noise is the term "transformers": in e-commerce networks, it usually refers to shape-changing toys, whereas in academic networks, it denotes deep learning models. Ignoring these types of noise during the architecture search process results in selecting architectures that fit the noise rather than capturing the underlying task-relevant information, ultimately leading to suboptimal GNN architectures.

In this paper, we investigate the problem of denoised graph neural architecture search in the presence of *structural and semantic noise* for the first time, providing comprehensive theoretical analyses and empirical evaluations. We first present a theoretical analysis of the mechanisms underlying GNAS methods for selecting optimal GNN architectures.

• *All authors were with the Department of Computer Science and Technology, BNRist, Tsinghua University, China, 100084. E-mail: {xin_wang, wwzhu}@tsinghua.edu.cn, {chb24,xlx21}@mails.tsinghua.edu.cn, zhangzey16@tsinghua.org.cn. Corresponding Author: Wenwu Zhu*

This work was supported by the National Key Research and Development Program of China No.2023YFF1205001, Beijing National Research Center for Information Science and Technology under Grant No.BNR2026TD03005.

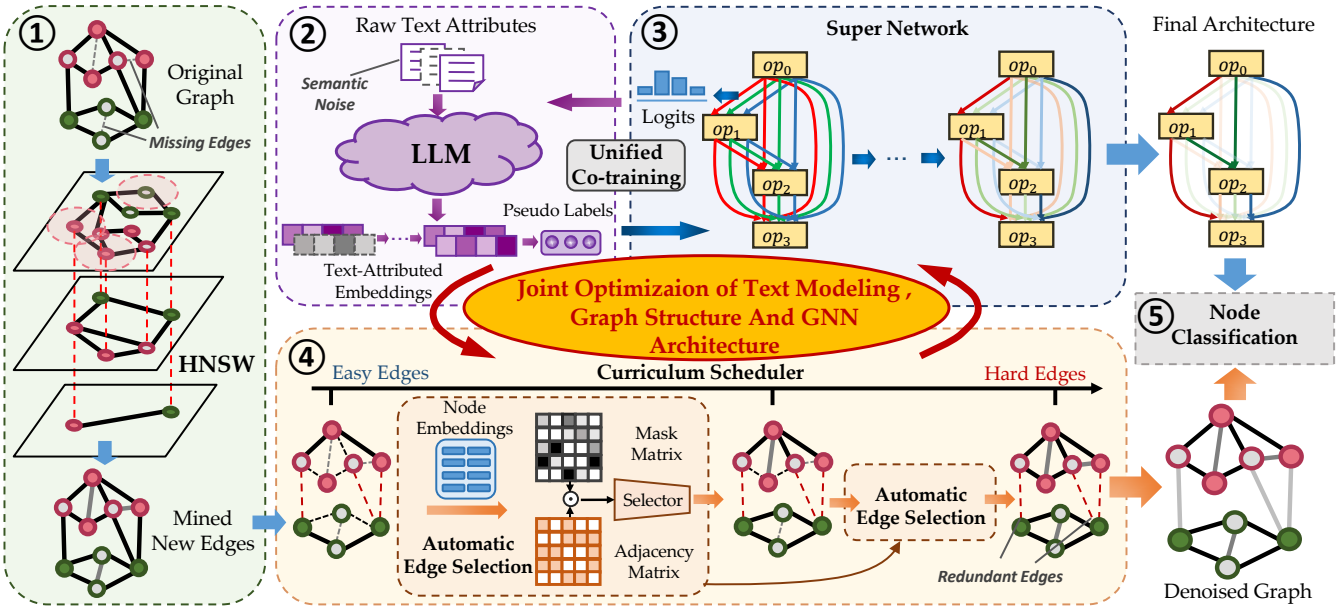


Fig. 1: An overview of the framework of Curriculum-GraphLLM, where the numbered steps correspond to the main workflow of denoised graph neural architecture search. Specifically, (1) a Hierarchical Navigable Small World (HNSW) graph is constructed to mine potential edges for alleviating missing-edge structural noise; (2) LLMs are employed to model text attributes and generate pseudo labels for unlabeled nodes, which are distilled to guide the optimization of the GNN supernet architecture; (3) the logits from the GNN are fed back into the LLM, forming a co-training framework that enhances text-attributed node embeddings and mitigates semantic noise from raw ambiguous text; (4) the resulting node embeddings are passed through a curriculum-scheduled automatic edge selection mechanism, which gradually involves edges in an easy-to-hard manner while excluding redundant ones during training; and (5) the optimized GNN architecture and denoised graph are utilized for downstream tasks such as node classification.

Then, we conduct synthetic experiments designed to systematically measure these mechanisms in practice. Our findings indicate that differentiable GNAS methods typically select architectures through effectively leveraging task-relevant information carried in the graph. However, our analyses further reveal that these methods are highly sensitive to *structural and semantic noise*, often leading to the selection of suboptimal GNN architectures.

Based on the above discoveries, we propose Curriculum-GraphLLM, a novel graphLLM framework for joint Optimization of architectures, structures, and texts for denoised graph neural architecture search. Curriculum-GraphLLM combines differentiable architecture search, dynamic topology updating, and semantic denoising into a unified framework. The graph structure adjustment serves as an adaptive noise reduction method to enhance architecture search performance. We first employ a differentiable optimization of the graph structure during the training process. Then, we further propose a scalable node-aware approximate nearest neighbor algorithm to mine potential connections between nodes not present in the original graph, and we use hidden feature smoothness to evaluate edge importance and constrain edge weights. We jointly optimize the parameters of the neural architecture and graph structure in a curriculum iterative updating manner. Moreover, to mitigate semantic noise on TAGs, we integrate LLMs as an auxiliary text modeling module rather than the central component of the framework, enabling text representations, graph structures, and GNN architectures to be co-optimized for denoised GNAS. The framework of Curriculum-GraphLLM is illustrated in Figure 1. We evaluate our proposed Curriculum-GraphLLM on several benchmark datasets. The results demonstrate that our model achieves consistently competitive or superior performance compared with existing baselines, especially under structural and semantic noise.

In summary, we make the following contributions:

- We present a theoretical analysis of the mechanism by which GNAS selects optimal GNN architectures. Specifically, we demonstrate for the first time that (i) differentiable graph architecture search tends to favor operations capable of correcting predictions on challenging data instances, and (ii) various operations are better suited for graphs depending on the levels of information contained in node features and graph structures.
- We design a measurement study with synthetic experiments to demonstrate that (i) differentiable graph architecture search can select operations based on the usefulness of the information in graphs, and (ii) noise in node features and graph structures can degrade the performance of architecture search.
- We propose Curriculum-GraphLLM, a framework for joint optimization of GNN architectures, graph structures, and textual semantics for denoised graph neural architecture search.
- Experimental results on several graph benchmark datasets demonstrate that Curriculum-GraphLLM achieves consistently competitive or superior performance compared with existing methods, especially under

structural and semantic noise.

This manuscript is an extension of our paper published at NeurIPS 2021 [17]. Compared to the conference version, we make new contributions in the following five aspects:

1) Our newly proposed Curriculum-GraphLLM is capable of introducing potential connections that are not present in the original graph through a node-aware approximate nearest neighbor algorithm, thereby enabling more effective optimization of the graph structure within the NAS procedure (please refer to Section 4.2).
2) The module of adding graph connections in Curriculum-GraphLLM can be easily integrated into the joint optimization scheme of architecture and graph structure while maintaining a relatively low time complexity ($O(|\mathcal{E}|log|\mathcal{V}|)$ (please refer to Section 4.5).
3) We propose a curriculum-based joint optimization scheme to gradually adapt the graph structure during the NAS process, along with a guarantee of convergence (please refer to Section 4.3).
4) We take text-attributed graphs (TAGs) into account and incorporate LLMs as an auxiliary semantic denoising module to jointly optimize text modeling, graph structure, and GNN architecture (please refer to Section 4.4).
5) We expand the experimental setup by including 11 additional graph datasets (8 real-world and 3 synthetic) and 20 additional baselines. The experimental results demonstrate that Curriculum-GraphLLM can mine real connections and achieve consistently competitive or superior performance compared to baselines.

2 Related Work

In this section, we review related works, including graph neural architecture search, graph structure learning, and text-attributed graph modeling with language models.

2.1 Graph Neural Architecture Search

Neural Architecture Search (NAS) has attracted increasing attention for its ability to automate the design of neural networks tailored to specific tasks. In particular, Graph Neural Architecture Search (GNAS) methods have been proposed to tackle the unique challenge of capturing the intricate relationships between neural architectures and complex graph-structured data [18], [19], [20], [21], [22], [23], [24]. Existing works mainly focus on defining a proper search space and search strategy for GNNs. These methods can be broadly classified into three categories: reinforcement-learning-based approaches [25], [26]; evolutionary-based strategies; and differentiable methods [27], [28], which enable continuous optimization of architectures within a differentiable search space. However, existing GNAS methods ignore the impact of structure noise on the search process, which can lead to suboptimal architectures, leading to suboptimal performance.

2.2 Graph Structure Learning

Graph structure learning is studied in recent GNN methods [29], [30] [31], [32], [33], [34], [35]. Since graph structure plays an important role in the GNN scheme, noise in the original graph can significantly harm GNNs. Graph structure learning aims to generate a clean graph during the training procedure by utilizing prior constraints, such as low rank [36] and smoothness [29]. Learning a new graph structure improves the model's ability to denoise, as well as its robustness to adversarial attacks on GNNs [37], [38], [39], [40], [41]. Existing graph structure learning works can be mainly divided into two parts [42]: graph regularization and graph structure modeling. Graph regularization performs structure learning and refines graphs from data with the original topology information, *e.g.*, sparsity, smoothness, and community preservation. Graph structure modeling uses metric-based kernels and neural networks to mine a clean graph structure without initial topology [43]. Attention-based methods can also be regarded as graph structure modeling [6], [44]. However, most existing methods focus on deleting or optimizing the weights of existing edges, ignoring the redundant edges.

2.3 Graph Curriculum Learning

Graph Curriculum Learning (GCL) differs from traditional Curriculum Learning due to the inherent dependencies within graph data [45], [46]. To address this, researchers leverage graph structures to assess difficulty through predefined or automated strategies. For example, CLNode [47], [48] is a GCL method that evaluates local difficulty by examining class diversity among a node's neighbors and uses global features to identify mislabeled nodes. RCL [49] incrementally integrates node relationships into the training process, adjusting for relationship complexity. However, adopting a curriculum learning strategy in the optimization of graph structure and architecture has not been explored [18], [20], [50], [51]. In this paper, we propose a curriculum-based joint optimization scheme to gradually adapt the graph structure during the GNAS process.

2.4 Text-Attributed Graph Modeling with Large Language Models

Large Language Models (LLMs) have demonstrated significant potential in enhancing the performance of GNNs by retrieving and incorporating relevant semantic knowledge into the graph representation of TAGs [52], [53], [54], [55], [56]. For instance, LLMs can retrieve domain-specific knowledge from external sources, which helps to refine the initial embeddings of graph nodes [55], [57]. Moreover, LLMs have the capability to encode entire graph structures through meticulously crafted prompts, allowing them to generate predictions in an autoregressive fashion [33], [54]. Another promising integration is the alignment of LLMs with GNNs in a shared vector space [51]. This alignment allows both models to operate within the same embedding framework, enabling GNNs to leverage the contextual knowledge embedded in LLMs. Despite the significant advancements in the field, current studies have overlooked the potential for a comprehensive and mutually beneficial integration between the text modeling abilities of LLMs and the mechanisms of neural architecture search and graph structure optimization.

3 Exploring the Neural Architecture Search Process for Graphs

In this section, we present the results of our exploratory experimental results conducted on differentiable GNAS.

3.1 Preliminary: Differentiable Architecture Search

We select DARTS [15] as the representative NAS method due to its conciseness and effectiveness. In DARTS, the neural architecture is encoded as a directed acyclic graph (DAG), with the input data serving as the source node and the edges representing the operations (such as GCN layers) applied to the data. DARTS utilizes a micro search space and focuses on identifying the optimal operation selection and node connections within the DAG while keeping the calculations inside each operation predefined.

DARTS consists of two distinct phases: the search phase and the evaluation phase. In the search phase, a super network (as shown in Figure 1) is constructed, where edges exist between every two nodes, *i.e.*, $e_{i,j}$ exists, $\forall 0 \leq i < j \leq N-1$. Each edge is a mixed operation that can be computed using the formula $e_{i,j}(\mathbf{x}_i) = \sum_{o \in \mathcal{O}} \frac{exp\alpha_{i,j}^{o}}{\sum_{o' \in \mathbf{O}} exp\alpha_{i,j}^{o'}} \cdot o(\mathbf{x}_i)$, where $\mathbf{x_i}$ denotes the output of node i, $\mathcal{O}$ represents the candidate operation set, and α corresponds to the learnable architecture parameters. After computing the mixed weighted sum, each node aggregates all input edges by $\mathbf{x}_j = \sum_{i<j} e_{i,j}(\mathbf{x}_i)$. This approach includes all possible operations at all possible positions in the super network. DARTS employs differentiable methods to optimize both the architecture parameters α and the operation parameters, denoted as $\mathbf{W}$, through a bi-level optimization scheme:

$$\begin{aligned} &\min_{\mathcal{A}} \mathcal{L}_{val}(\mathbf{W}^*, \mathcal{A}) \\ s.t. \quad &\mathbf{W}^* = \arg\min_{\mathbf{W}} \mathbb{E}_{\mathcal{A} \in \Gamma(\mathcal{A})} \mathcal{L}_{train}(\mathbf{W}, \mathcal{A}), \end{aligned} \tag{1}$$

where $\Gamma(\mathcal{A})$ represents the architecture distribution learned in the search phase. By independently learning $\mathbf{W}$ and $\mathcal{A}$, the search efficiency is improved. Furthermore, since the parameters are trained in the super network, the parameters of different architectures are shared, leading to a reduction in the overall number of parameters.

Upon completion of the training process, the operation with the maximum α value is selected, and these operations constitute the optimal architecture.

3.2 Theoretical Analysis of Architecture Parameters

Next, we aim to investigate the behavior of DARTS in graph tasks and address the following two research questions: (i) *How does GNAS select its desired architectures?* and (ii) *To what extent are the architectures selected by GNAS optimal?*

We first provide a theoretical analysis of (i). The mixed operation is a crucial component in DARTS, with architecture parameters controlling the selection of operations. To explore how these architecture parameters evolve during the training process, we first analyze a simple case of a binary classification problem. In this scenario, there is only one layer within the super network, which contains two candidate operations requiring exploration. The mixed operation is defined as:

$$F(\mathbf{x}_k) = o_1 f_1(\mathbf{x}_k) + o_2 f_2(\mathbf{x}_k), \tag{2}$$

where $o_1 = \frac{exp\{\alpha_1\}}{exp\{\alpha_1\}+exp\{\alpha_2\}}$ and $o_2 = \frac{exp\{\alpha_2\}}{exp\{\alpha_1\}+exp\{\alpha_2\}}$, and f_1 and f_2 are candidate operations. We use gradient methods to minimize the following binary cross-entropy loss:

$$\mathcal{H}(F) = -\sum_k \Big[y_k \log\Big(\sigma\big(F(\mathbf{x}_k)\big)\Big) + (1-y_k) \log\Big(1-\sigma\big(F(\mathbf{x}_k)\big)\Big)\Big], \tag{3}$$

where $\sigma(\cdot)$ is the Sigmoid function and $y_k \in \{0,1\}$ is the ground truth label of data $\mathbf{x}_k$. We present the following theorem along with a proof.

Theorem 1. The changes in the operation weights o_n after one step of gradient descent with respect to architecture parameters are dependent on the score $\sum_k \big[w_k f_n(\mathbf{x}_k)\big]$, where $w_k = \sigma\big(F(\mathbf{x}_k)\big) - y_k$. Specifically, if $\sum_k w_k f_1(\mathbf{x}_k) < \sum_k w_k f_2(\mathbf{x}_k)$, then o_1 increases and o_2 decreases, and vice versa.

Proof 3.1.

We provide the proof for Theorem 1, beginning with Lemma 1:

Lemma 1. The operation weights are calculated by a softmax function. *e.g.*, $o_1 = \frac{exp\{\alpha_1\}}{exp\{\alpha_1\}+exp\{\alpha_2\}}$. Let g_1 and g_2 are the gradients w.r.t. α_1 and α_2. The operation weights after a gradient descent step are o_1' and o_2', *e.g.*, $o_1' = \frac{exp\{\alpha_1-g_1\}}{exp\{\alpha_1-g_1\}+exp\{\alpha_2-g_2\}}$. If $g_1 < g_2$, then $o_1' > o_1$ and $o_2' < o_2$, and vice versa.

Proof of Lemma 1:
If $g_1 < g_2$, we should improve $o_1' > o_1$, that is:

$$\begin{aligned} \frac{exp\{\alpha_1 - g_1\}}{exp\{\alpha_1 - g_1\} + exp\{\alpha_2 - g_2\}} &> \frac{exp\{\alpha_1\}}{exp\{\alpha_1\} + exp\{\alpha_2\}} \\ exp\{\alpha_1 - g_1\}\big(exp\{\alpha_1\} + exp\{\alpha_2\}\big) &> exp\{\alpha_1\}\big(exp\{\alpha_1 - g_1\} \\ &\quad + exp\{\alpha_2 - g_2\}\big) \\ exp\{2\alpha_1 - g_1\} + exp\{\alpha_1 + \alpha_2 - g_1\} &> exp\{2\alpha_1 - g_1\} \\ &\quad + exp\{\alpha_1 + \alpha_2 - g_2\} \\ exp\{\alpha_1 + \alpha_2 - g_1\} &> exp\{\alpha_1 + \alpha_2 - g_2\} \\ \alpha_1 + \alpha_2 - g_1 &> \alpha_1 + \alpha_2 - g_2 \\ g_1 &< g_2. \end{aligned} \tag{4}$$

We already have $g_1 < g_2$, thus $o_1' > o_1$. Since $o_1 + o_2 = o_1' + o_2' = 1$, we have $o_2' < o_2$, which finishes the proof of Lemma 1.
Then we prove Theorem 1 with Lemma 1:
The binary cross-entropy loss is:

$$\begin{aligned} H(F) &= -\sum_k \Big[y_k \cdot ln\sigma\big(-F(\mathbf{x}_k)\big) + (1-y_k) \cdot ln\Big(1-\sigma\big(-F(\mathbf{x}_k)\big)\Big)\Big] \\ &= -\sum_k \Big[y_k \cdot \Big(-ln\big(1+exp\{-F(\mathbf{x}_k)\}\big)\Big) \\ &\quad + (1-y_k) \cdot \Big(-F(\mathbf{x}_k) - ln\big(1+exp\{-F(\mathbf{x}_k)\}\big)\Big)\Big] \\ &= \sum_k \Big[ln\big(1+exp\{-F(\mathbf{x}_k)\}\big) + (1-y_k) \cdot F(\mathbf{x}_k)\Big]. \end{aligned} \tag{5}$$

Then we have:

$$\begin{aligned} \frac{\partial H(F)}{\partial o_i} &= \sum_k \Big[\frac{-exp\{-F(\mathbf{x}_k)\}}{1+exp\{-F(\mathbf{x}_k)\}} \cdot f_i(\mathbf{x}_k) + (1-y_k) \cdot f_i(\mathbf{x}_k)\Big] \\ &= \sum_k \Big[\Big(\sigma\big(F(\mathbf{x}_k)\big) - y_k\Big) \cdot f_i(\mathbf{x}_k)\Big] \\ &= \sum_k \Big[w_k \cdot f_i(\mathbf{x}_k)\Big]. \end{aligned} \tag{6}$$

Besides, since $o_1 = \frac{exp\{\alpha_1\}}{exp\{\alpha_1\}+exp\{\alpha_2\}}$, we have:

$$\begin{aligned}\frac{\partial o_1}{\partial \alpha_1} &= \frac{exp\{\alpha_1\}\cdot\Big(exp\{\alpha_1\}+exp\{\alpha_2\}\Big) - exp\{\alpha_1\}\cdot exp\{\alpha_1\}}{\Big(exp\{\alpha_1\}+exp\{\alpha_2\}\Big)^2} \\ &= \frac{exp\{\alpha_1+\alpha_2\}}{\Big(exp\{\alpha_1\}+exp\{\alpha_2\}\Big)^2}.\end{aligned} \tag{7}$$

Similarly:

$$\frac{\partial o_2}{\partial \alpha_2} = \frac{exp\{\alpha_1+\alpha_2\}}{\Big(exp\{\alpha_1\}+exp\{\alpha_2\}\Big)^2}. \tag{8}$$

Thus $\frac{\partial o_1}{\partial \alpha_1} = \frac{\partial o_2}{\partial \alpha_2}$. If we have the relationship between the two scores:

$$\begin{aligned}\sum_k \big[w_k f_1(\mathbf{x}_k)\big] &< \sum_k \big[w_k f_2(\mathbf{x}_k)\big] \\ \frac{\partial H(F)}{\partial o_1} &< \frac{\partial H(F)}{\partial o_2} \\ \frac{\partial H(F)}{\partial o_1}\frac{\partial o_1}{\partial \alpha_1} &< \frac{\partial H(F)}{\partial o_2}\frac{\partial o_2}{\partial \alpha_2} \\ \nabla_{\alpha_1} &< \nabla_{\alpha_2}.\end{aligned} \tag{9}$$

By Lemma 1, o_1 increases and o_2 decreases.

Theorem 1 shows that DARTS evaluates various candidate operations by employing a scoring mechanism, which can be interpreted as a weighted sum of logits. The weights, denoted by $w_k \in (-1, 1)$, are positive or negative based on the data label. If $y_k = 0$, the weight is a positive number $\sigma(F(\mathbf{x}_k))$, indicating that those data with poor predictions given by the super network have larger weights. By focusing exclusively on scores associated with positive weights, the candidate operation that provides improved predictions will exhibit smaller logits. The sum of these scores yields a lower score and increases the architecture weight o_n. A similar conclusion can be drawn by analyzing data with label $y_k = 1$. In summary, DARTS favors operations that can improve predictions on difficult data.

3.3 Analyses on Synthetic Graphs

In this subsection, we investigate the behavior of DARTS on graph tasks using synthetic graphs to prove existing GNAS methods are sensitive to *structural and semantic noise* in graph data.

3.3.1 Synthetic Graph Generation

We construct a synthetic graph comprising 600 nodes with 10 types, with 60 nodes per type. We focus on the node classification task where the node type serves as the node label. The node features of a particular type i follow a Gaussian distribution, denoted by $\mathcal{N}(\mu_i, \sigma_1^2)$ (for convenience in notations, we abbreviate the feature as one-dimensional, while the features can compose of multiple independent channels), where μ_i is sampled from another Gaussian distribution, denoted by $\mathcal{N}(0, \sigma_2^2)$. Here, we use $\beta = \frac{\sigma_2}{\sigma_1}$ to measure and control the difficulty of classifying nodes based on their features, *i.e.*, if β is large, intra-group node features are closely clustered while inter-group node features are further apart, making it easier to classify nodes based on their features, and vice versa. This difficulty measurement also reflects the ratio of *semantic noise* to type-relevant information within the node features.

As for the edges, we set two hyperparameters, p_1 and p_2, where p_1 is the probability of generating an edge between two intra-group nodes, and p_2 is the probability of generating an edge between two inter-group nodes. We further define $\delta = \frac{p_1}{p_2}$, which represents the difficulty of classifying nodes by considering their neighborhoods. If δ is large, intra-group nodes are more connected in the neighborhood, making it easier to classify nodes by aggregating neighborhood information, and vice versa. This difficulty measurement also reflects the ratio of *structural noise* to type-relevant information within the graph structure.

We generate synthetic graphs with different β and δ, and split the nodes under a semi-supervised setting for the purpose of performing the node classification task.

3.3.2 Theoretical Analysis

To explore the behavior of different operations under different settings of the synthetic graph, we theoretically analyze a simple scenario in this subsection. We consider only one target node connected to two classes of nodes, where one class is the same as the target node's class, and the other is not. The node feature channel is set as 1, and the candidate operations consist of only *linear* and *GCN* [5], which represent two types of operations: message-passing and non-message-passing operations. We have the following result.

Theorem 2. Under our synthetic graph setting, let n denote the number of edges connecting the target node, and $|D|$ follows a Gaussian distribution $\mathcal{N}(0, \beta^2)$, which measures the relative distance between the centers of the two classes. Then, the probability that the linear operation provides more accurate predictions than GCN on the target node is given by $P = \Phi(\frac{\sqrt{2}n|D|}{(\delta+1)\sqrt{(n+1)(n+2)}})$, where Φ denotes the cumulative distribution function of the standard Gaussian distribution.

Proof 3.2.

Firstly, we have Property 1, the proof of Property 1 can be easily found in any classic Probability Theory textbook.

Property 1. If $x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$, $x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$, then $x_1 + x_2 \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$

Since the centers of the two classes follow $\mu_1, \mu_2 \sim \mathcal{N}(0, \sigma_2^2)$, according to Property 1, the absolute distance between these two centers is $d \sim \mathcal{N}(0, 2\sigma_2^2)$. Since $D \sim \mathcal{N}(0, \beta^2)$ and $\beta = \frac{\sigma_2}{\sigma_1}$, we know the relationship between D and d is $\frac{d}{D} = \sqrt{2}\sigma_1$. We assume the center of the target node's class is μ, and the center of the other class is $\mu + |d|$. Thus the target node's feature follows $x_t \sim \mathcal{N}(\mu, \sigma_1)$. The node feature of the other class follows $x \sim \mathcal{N}(\mu + |d|, \sigma_1)$.

GCN can be seen as a message-passing process followed by a linear layer, where the message-passing process is an average of all node neighbors. The number of the target node's intra-group neighbors is $\frac{\delta n}{\delta+1} + 1$ (include self-loop), while the number of the target node's inter-group neighbors is $\frac{n}{\delta+1}$. Let the probability of a target node's neighbor becoming an inter-group neighbor be p, defined as follows:

$$p = \frac{\frac{n}{\delta+1}}{\frac{\delta n}{\delta+1} + 1 + \frac{n}{\delta+1}} = \frac{n}{(\delta+1)(n+1)}. \tag{10}$$

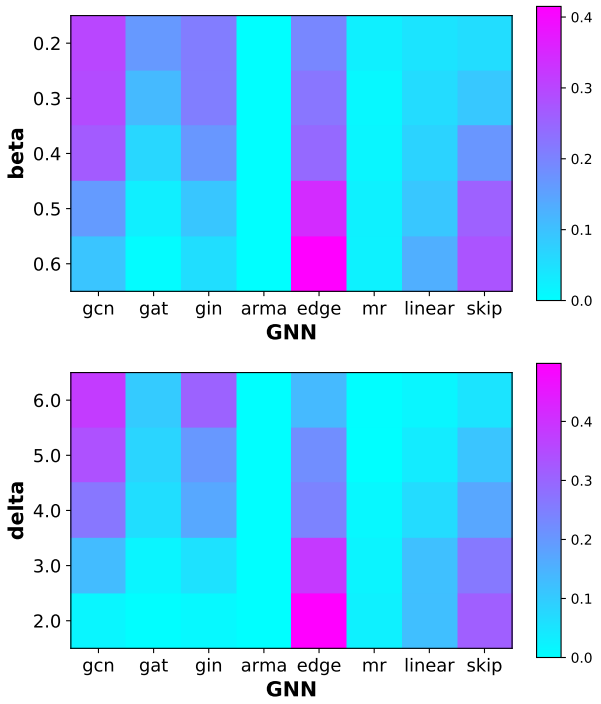


Fig. 2: The frequency of GNN operations in optimal architectures on different graph datasets.

Let x_G represent the node feature after the message-passing process. We have $x_G \sim \mathcal{N}(\mu + p|d|, \frac{\sigma_1^2}{n+1})$. Linear operation giving a more accurate prediction than GCN on the target node is equivalent to $x_t < x_G$. Let $\Delta = x_t - x_G$, we have $\Delta \sim \mathcal{N}(-p|d|, \frac{n+2}{n+1}\sigma_1^2)$. Thus we have:

$$\begin{aligned} P(x_t < x_G) &= P(\Delta < 0) \\ &= \Phi\Big(\frac{p \cdot |d|}{\sqrt{\frac{n+2}{n+1}}\sigma_1}\Big) \\ &= \Phi\Big(\frac{\frac{n}{(\delta+1)(n+1)} \cdot \sqrt{2}\sigma_1|D|}{\sqrt{\frac{n+2}{n+1}}\sigma_1}\Big) \\ &= \Phi\Big(\frac{\sqrt{2}n|D|}{(\delta+1)\sqrt{(n+1)(n+2)}}\Big). \end{aligned} \tag{11}$$

Theorem 2 demonstrates that the performance of different operations is influenced by β, δ, and n. Specifically, as β increases, the expectation of $|D|$ increases, and the probability of selecting *linear* increases. On the other hand, as δ increases, the probability of selecting *GCN* increases. The number of edges can also affect the probability, but to a lesser extent than β and δ.

Next, we report experimental results on synthetic graphs.

3.3.3 Operation Selection

To investigate how β and δ influence DARTS in practice, we generate synthetic graphs by controlling one variable while changing the other. Specifically, we set $\beta = 0.4$ and vary δ, and set $\delta = 4.0$ and vary β. For each setting of β and δ, we generate 100 different graphs and apply DARTS to them. We choose *GCN* [5], *GAT* [6], *GIN* [7], *MRConv* [58], *EdgeConv* [59], *linear*, *skip connect* and *zero* as candidate operations in DARTS, with the number of operations set to 6. We focus on exploring the behavior of DARTS in operation selection rather than connection design. We count the frequency of each operation that appears in the optimal architecture and present the results in Figure 2.

Based on the results, we observe that DARTS suggests using different operations for different graphs. Specifically, for graphs with more structural information (*i.e.*, large δ) and less feature information (*i.e.*, small β), DARTS tends to select typical message-passing operations, such as GIN. When there is less structural information (*i.e.*, small δ) and more feature information (*i.e.*, large β), DARTS is more likely to select operations like *linear* and *skip connect* to avoid message passing. These results are consistent with Theorem 2.

By integrating the two theorems with the experimental results, we provide an analysis addressing question (i), specifically elucidating the mechanism by which GNAS selects its optimal architectures. A GNN can be considered as an advanced information extraction mechanism that aggregates valuable messages from neighboring nodes, thereby enabling precise node classification. The operations within the search space possess varying capacities for extracting meaningful information from both node features and graph structures. For example, *GCN* can extract information from the graph structure but reduce the influence of node features, while *linear* only uses feature information and ignores the graph structure. DARTS aims to find the optimal information extractor to improve the prediction of the super network by assessing the usefulness of node features and graph structure for the task. If the graph structure is useful, DARTS will select operations that extract information from the topology. If node features are useful, DARTS will prevent message passing to focus on the features.

Furthermore, we find that the heat maps of the two experimental results of varying β and δ are similar, indicating that DARTS judges the relative information of the two types rather than the absolute amount. This is also consistent with Theorem 2. For example, as long as node features are more useful than graph structure, the structure is considered to have certain noise that is harmful to the task, and message-passing is preferred to be avoided.

We have demonstrated that DARTS is capable of evaluating the relevance of various graph information sources and selecting suitable operations. This capability motivates us to investigate the denoising potential of DARTS. When a clean graph is perturbed by random *structural and semantic noise*, DARTS can re-evaluate the significance of graph structure and node features, thereby proposing new architecture designs as needed. However, our experiments on real graph datasets (see Section 5 for details) show that the denoising ability of DARTS is unstable. Therefore, we continue to explore how accurately DARTS selects operations.

3.3.4 Accuracy of Operation Selection

To further answer the question (ii), *i.e.*, to what extent are the architectures selected by GNAS optimal, we transfer the searched architectures to different graphs for evaluation. We vary β and δ by controlling one variable while changing the other. The results are presented in Table 1.

The conventional assumption holds that an architecture searched on a specific graph should exhibit optimal

TABLE 1: The GNN performance under different settings of β and δ for the searching and evaluation phase. "S" indicates the search phase, and "E" indicates the evaluation phase. All the results are an average of 100 runs with the variance reported after $\pm$ signs.

S \ E	$\beta = 0.2$	$\beta = 0.3$	$\beta = 0.4$	$\beta = 0.5$	$\beta = 0.6$
$\beta = 0.2$	$\mathbf{56.4}_{\pm 4.9}$	$70.9_{\pm 4.9}$	$82.0_{\pm 5.1}$	$87.9_{\pm 5.0}$	$91.6_{\pm 4.9}$
$\beta = 0.3$	$55.5_{\pm 5.1}$	$\mathbf{73.1}_{\pm 4.4}$	$85.1_{\pm 5.2}$	$91.7_{\pm 5.1}$	$94.4_{\pm 4.6}$
$\beta = 0.4$	$52.3_{\pm 5.6}$	$73.1_{\pm 4.8}$	$86.5_{\pm 5.4}$	$93.0_{\pm 5.0}$	$96.3_{\pm 4.2}$
$\beta = 0.5$	$48.5_{\pm 7.1}$	$72.7_{\pm 5.5}$	$\mathbf{88.6}_{\pm 3.1}$	$96.1_{\pm 1.8}$	$98.6_{\pm 1.2}$
$\beta = 0.6$	$43.0_{\pm 6.4}$	$69.4_{\pm 5.8}$	$88.1_{\pm 3.2}$	$\mathbf{96.3}_{\pm 1.6}$	$\mathbf{99.0}_{\pm 1.0}$

S \ E	$\delta = 6.0$	$\delta = 5.0$	$\delta = 4.0$	$\delta = 3.0$	$\delta = 2.0$
$\delta = 6.0$	$93.4_{\pm 2.2}$	$88.4_{\pm 3.0}$	$80.6_{\pm 4.4}$	$68.9_{\pm 6.7}$	$55.1_{\pm 8.1}$
$\delta = 5.0$	$94.2_{\pm 2.1}$	$90.1_{\pm 3.5}$	$83.6_{\pm 5.5}$	$79.2_{\pm 8.4}$	$62.9_{\pm 11.5}$
$\delta = 4.0$	$\mathbf{94.2}_{\pm 2.1}$	$\mathbf{91.3}_{\pm 3.0}$	$86.3_{\pm 5.2}$	$79.2_{\pm 8.4}$	$70.5_{\pm 12.4}$
$\delta = 3.0$	$92.9_{\pm 4.1}$	$91.0_{\pm 3.3}$	$\mathbf{88.7}_{\pm 3.3}$	$\mathbf{85.8}_{\pm 3.4}$	$82.1_{\pm 5.5}$
$\delta = 2.0$	$86.8_{\pm 4.6}$	$86.3_{\pm 4.6}$	$85.8_{\pm 4.4}$	$85.4_{\pm 3.6}$	$\mathbf{84.8}_{\pm 3.4}$

performance on that same graph, meaning the elements along the main diagonal in the tables should represent the highest values in each respective column. Nevertheless, our observations reveal that an architecture derived from a different graph can, in certain cases, achieve superior performance. For example, the best architecture evaluated on the graph with $\delta = 5$ is searched from the graph with $\delta = 4$. This phenomenon can provide some answers to question (ii), *i.e.*, DARTS may not be able to accurately select the optimal architecture for the node classification task.

Next, we will try to provide plausible explanations for this empirical observation. Although DARTS can assess the usefulness of information contained in the graph structure and node features, it may not always obtain an accurate result. For instance, when $\delta = 6$, the graph structure contains useful information for node classification, but DARTS may become overconfident and make incorrect decisions by aggregating too much information from the neighborhood, which includes *structural and semantic noise*. On the other hand, when there is less graph structure information (*e.g.*, $\delta = 4, 5$), DARTS becomes more conservative about aggregation. Therefore, architectures searched at these settings perform better when evaluated at $\delta = 6$. Similarly, DARTS may misjudge the usefulness of information in other settings.

Based on the above exploration, we have found a major cause for why DARTS may not design the optimal architecture: *structural and semantic noise* in the original graph. The message-passing scheme used in GNNs smooths node features, which can be viewed as a denoising process. Therefore, we should consider denoising the graph structure in the GNAS process. Combining architecture search and graph structure learning becomes a natural way to address this issue. In the next section, we will introduce our method that jointly optimizes neural architecture and graph structure.

4 The Proposed Method: Curriculum-GraphLLM

In this section, we present our proposed method in detail. First, we introduce dynamic topology learning through two sections: (1) We address the impact of noisy edges by introducing differentiable graph structure learning in Section 4.1. (2) To identify potentially useful edges that are missing in the original graph, we introduce a node-aware approximate nearest neighbor search algorithm in Section 4.2. In Section 4.3, we present the curriculum optimization framework which jointly optimizes the neural architecture and graph structure. Moreover, we further introduce a unified co-training framework in Section 4.4 to simultaneously optimize the text modeling, neural architecture, and graph structure. We analyze the time complexity and memory cost of our method in Section 4.5.

4.1 Structural Denoising: Removing Noisy Edges

4.1.1 Differentiable Graph Structure Learning

In most real graph datasets, the graph structure is discrete, *i.e.*, the edges given in the graph are unweighted. Although these edges contain crucial information, they also introduce certain noise into the data. For instance, in the citation dataset, a paper cites other papers for various reasons, and some citations are not helpful for the paper classification task. Similar phenomena can be observed in other types of graphs. Moreover, the discrete nature of graph structure makes it challenging to optimize the structure differently.

We apply a differentiable graph structure to address this issue. Specifically, we use a learnable parameter to represent the weight of each edge and enforce the weights to be in the range of $(0, 1)$ during message passing by using a sigmoid function. Thus, the normalized parameter matrix used in message passing can be represented as:

$$\tilde{\mathbf{G}}_{i,j} = \frac{\sigma(\mathbf{G}_{i,j})}{\sum_{j'} \sigma(\mathbf{G}_{i,j'})}, \tag{12}$$

where $\mathbf{G}_{i,j}$ and $\tilde{\mathbf{G}}_{i,j}$ is the unnormalized and normalized weight associated with edge (i, j), respectively. Note that we also adopt self-loops and fix the weights of self-loops. In the message-passing procedure, edge weights are multiplied before aggregation.

Differentiable graph structure brings two benefits. First, we can distinguish the usefulness of edges and aggregate more useful information during message passing. Second, by relaxing the discrete edges into continuous edge weights, we can use differentiable methods to optimize the structure, *i.e.*, $\mathbf{G}_{i,j}$. Note that this differentiable graph structure is compatible with most GNNs by simply replacing the adjacent matrix $\mathbf{A}$ with the parameter matrix $\tilde{\mathbf{G}}$.

4.1.2 Hidden Feature Smoothness Regularizer

To regularize the learning of graph structure, we further utilize hidden node features $\mathbf{H}$ to optimize the parameter matrix $\mathbf{G}$. The fundamental assumption is that nodes with similar features should be connected, which is also known as the homophily assumption [60] or the first-order proximity between nodes [61]. Specifically, we use the following hidden feature smoothness regularizer, which is defined as the weighted sum of the Euclidean distance of the hidden feature between connected nodes:

$$\mathcal{L}_1 = \sum_{(i,j)\in\mathcal{E}} \tilde{\mathbf{G}}_{i,j} \left\| \mathbf{h}_i - \mathbf{h}_j \right\|_2, \tag{13}$$

where $\mathbf{h}_i$ is the hidden representation of node i and N denotes the number of nodes. With this regularizer, the edge weights of nodes with similar hidden features are larger than those of nodes with different hidden features. By learning the graph structure, nodes with similar features are aggregated

more in the message-passing process, which can improve the denoising ability of the model.

Meanwhile, we also aim to prevent the learned graph structure from deviating excessively from the original graph. Consequently, we incorporate a distance regularizer to measure and constrain the discrepancy between the learned graph structure and the original graph structure. The overall loss of graph structure learning is formulated as:

$$\mathcal{L}_s = \lambda_1 \mathcal{L}_{\text{cls}} + \lambda_2 \sum_{(i,j)\in\mathcal{E}} \tilde{\mathbf{G}}_{i,j} \left\| \mathbf{h}_i - \mathbf{h}_j \right\|_2 + \sum_{(i,j)\in\mathcal{E}} (\tilde{\mathbf{G}}_{i,j} - \mathbf{A}_{i,j})^2, \tag{14}$$

where $\mathcal{E}$ is the potential edge set, $\mathcal{L}_{\text{cls}}$ is the task loss, $\mathbf{A}_{i,j}$ is the adjacency matrix, and λ_1, λ_2 are hyper-parameters to control the weights of different objectives.

4.2 Structural Denoising: Adding Potential Edges

In Section 4.1, we focus on the removal of noisy edges from the original graph. In this section, we present our method for adding potential edges that are not present in the original graph. Notice that considering all possible edges between every pair of nodes would result in a time complexity of $O(|\mathcal{N}|^2)$, where $(|\mathcal{N}|$ denotes the number of nodes, which is impractical for large-scale graphs. To address this issue, we introduce a scalable approach for adding potential edges. The core idea is to only consider the nearest neighbors of each node as potential edges and efficiently find the nearest neighbors using Hierarchical Navigable Small World (HNSW) [62] graphs.

4.2.1 HNSW Graph Construction

HNSW is an effective algorithm for Approximate Nearest Neighbors (ANN). The ANN problem is a faster approximation of the traditional K-Nearest Neighbor (KNN) problem, with a small margin of approximation error. Compared to KNN methods, ANN algorithms have improved scalability.

In HNSW, we transform the original graph data into a series of hierarchical small-world graphs, where the upper graphs are coarser than the lower graphs, indicating that the nodes in the upper graphs are a subset of the nodes in the lower graphs. The lowest graph contains all nodes. The nodes are sequentially inserted into the hierarchical graphs, with an exponentially decaying probability to determine the layer into which each node is inserted. Specifically, a probability parameter $p \in (0, 1)$ is chosen, and the probability of a node being inserted in the l^{th} layer is p^{l-1}. The inserted node establishes d connections with its approximate nearest neighbors in each small-world graph. Additionally, every node is allowed to have at most $d_{\max}$ connections in each small-world graph.

When inserting a new node, we need to add edges for the node in HNSW, similarly to normal ANN searches. Specifically, we begin at the top graph and greedily traverse the graph until a local minimum is reached, *i.e.*, the node closest to the target node in the top graph is found. Then, the algorithm switches to a layer below, where the previously found local minimum node becomes the starting node. The algorithm traverses the graph to find a local minimum again. In this new search process, since we already know the starting node is the local minimum in the last layer, the algorithm does not need to consider nodes belonging to the previous layer. The algorithm repeats this process until reaching the bottom layer, where approximate nearest neighbors are obtained. Since the maximum number of edges per node in all layers is restricted to a constant number, the searching algorithm only has a logarithmic complexity. We refer readers to for more details about the HNSW graph construction.

4.2.2 Node-aware Approximate Nearest Neighbor Search

After constructing the HNSW graph, we need to conduct an approximate nearest neighbor search to identify potential edges that could be added to the original graph. However, using the original HNSW search algorithm to find the nearest node pairs ignores the characteristics of different nodes since it treats all nodes equally. Certain nodes may have a higher probability of having connections based on their specific characteristics. For example, in citation graphs, some papers have general contributions, while others provide minor improvements. Although they have similar topics, the former is more likely to be cited by later studies. We identify this characteristic in the original graph and propose a node-aware approximate nearest neighbor algorithm.

Algorithm 1 Node-aware Approximate Nearest Neighbor Search

Require: The node set $\mathcal{V}$, the node degrees in original graph $\{d_q | q \in \mathcal{V}\}$, the normalization factor for level generation m_L.
Ensure: Node-aware approximate nearest neighbor pairs.

Sort $\mathcal{V}$ according to the degree of nodes in the original graph from largest to smallest
$ep = \varnothing$ # the enter point of HNSW
$L = -1$ # the level of HNSW
for $q \in \mathcal{V}$ **do**
  $\mathcal{V}_n = \varnothing$ # nearest elements
  $l = \lfloor -ln\big(U(0,1)\big) \cdot m_L \rfloor$ # level of q
  if $l > L$ **then**
    Construct graph $(l+1) \dots L$ with single element q
    $L = l$ # the level of HNSW increases
    $ep = q$ # set enter point as q
  else
    $np = ep$ # start from the enter point
    for $l_c = L \dots (l+1)$ **do**
      $np =$ the nearest elements to q at layer l_c
    end for
  end if
  for $l_c = l \dots 0$ **do**
    $\mathcal{V}_n =$ the d_q nearest elements to q at layer l_c
    Add connections from $\mathcal{V}_n$ to q at layer l_c
    for $v \in \mathcal{V}_n$ **do**
      if v has more than $d_v * 2$ connections at layer l_c **then**
        Only keep $d_v * 2$ nearest neighbors to v
      end if
    end for
    $np =$ the nearest point in $\mathcal{V}_n$
  end for
end for
Construct $\mathcal{E}'$ by combining all edges at layer 0
Return $\mathcal{E}'$

Specifically, our method improves on the original algorithm in two ways. Firstly, since the node inserted first has a higher probability of obtaining more connections in small-world graphs, we insert nodes based on their original degree, from largest to smallest, instead of random order in traditional HNSW graphs. Secondly, we use customized $d_{\max}$

Algorithm 2 GASSIO

Require: Graph feature matrix $\mathbf{X}$, the adjacent matrix $\mathbf{A}$, the learning rates η_W, $\eta_{\mathcal{A}}$ and η_G, the number of warming up epoches n_w.

Initialize the super network F with parameters $\mathbf{W}$, $\mathcal{A}$
Initialize the parameter matrix $\mathbf{G}$
Initialize $\mathbf{A}^0 = \mathbf{A}$, $\mathbf{S}^0$
while not converge **do**
  $\mathbf{W} = \mathbf{W} - \eta_W \nabla_W \mathcal{L}_{\text{train}}(\mathbf{W}, \mathcal{A}, \mathbf{G}, \mathbf{A}^t) + \frac{\gamma}{2} \left\| \mathbf{W} - \mathbf{W}^{t-1} \right\|_2^2$
  if $epoch < n_w$ **then**
    Continue
  else if $epoch = n_w$ **then**
    Obtain $\mathcal{E}'$ by Algorithm 1
    Add edges in $\mathcal{E}'$ to $\mathbf{G}$
  else if $epoch > n_w$ **then**
    Calculate $\mathbf{S}^t$ in Eq (15)
    Calculate $\mathbf{A}^t = \mathbf{S}^t \odot \mathbf{A}$
  end if
  $\mathbf{G} = \mathbf{G} - \eta_G \nabla_{\mathbf{G}} \mathcal{L}_s(\mathbf{W}, \mathcal{A}, \mathbf{G}, \mathbf{A}^t)$
  $\mathcal{A} = \mathcal{A} - \eta_{\mathcal{A}} \nabla_{\mathcal{A}} \mathcal{L}_{\text{val}}(\mathbf{W}, \mathcal{A}, \mathbf{G}, \mathbf{A}^t)$
end while
Return $f_{\mathbf{W},\mathcal{A},\mathbf{G}}(X)$

values for different nodes. For a node i, we set its number of connections established at HNSW graph construction to be twice the node's original degree. As a result, the maximum possible degree for a node after adding potential edges is three times the node's original degree. Consequently, nodes have different numbers of edges in the hierarchical small-world graphs, which is consistent with their characteristics in the original graph.

After all nodes are inserted, the small world graph in the bottom layer will have all candidate ANN pairs, so we take out the edges in the bottom graph as the potential edges, denoted as $\mathcal{E}'$. The overall algorithm of our proposed node-aware approximate nearest neighbor search is shown in Algorithm 1. Then, we modify the graph structure learning loss in Eq (14) by replacing $\mathcal{E}$ with $\mathcal{E} \bigcup \mathcal{E}'$.

4.3 Structural Denoising: Curriculum Optimization of Neural Architecture and Structure

Curriculum optimization plays a vital role in enabling models to learn in a staged, progressive manner. Yet, current approaches have overlooked its potential in simultaneously optimizing neural architectures and graph structures. Here, we present a strategy that leverages curriculum optimization to address this gap, promoting the joint refinement of both components.

4.3.1 Automatic Edge Selection for Joint Optimization

We begin by measuring the difficulty of each edge in the graph using the normalized weight, $\tilde{\mathbf{G}}_{i,j}$, associated with edge (i, j) as defined in Eq (12). A lower weight for the edge (i, j) indicates a higher likelihood that the current architecture expects this edge, suggesting that the edge is easier to learn and less likely to contain noise.

To facilitate the automatic selection of additional edges for joint optimization of the neural architecture and graph structure, we propose an automatic edge selection strategy. Specifically, we introduce a learnable mask matrix $\mathbf{S}$ to filter out edges, where each element $\mathbf{S}_{i,j}$ lies in the interval $[0, 1]$. We penalize the normalized weight $\tilde{\mathbf{G}}_{i,j}$ of the selected edges using this mask matrix $\mathbf{S}$, which can be expressed as $\sum_{(i,j)\in\mathcal{E}} \mathbf{S}_{i,j} \tilde{\mathbf{G}}_{i,j}$. To control the number of selected edges, we incorporate a regularization term $\sum_{(i,j)\in\mathcal{E}} \mathbf{S}_{i,j}$ into the loss function. Additionally, we introduce a proximal term $\frac{\gamma}{2} \left\| \mathbf{S} - \mathbf{S}^{t-1} \right\|_2^2$ to control the magnitude of update. The resulting overall loss function for selecting edges $\mathbf{S}^t$ at step t is formulated as:

$$\begin{aligned} &\min_{\mathbf{S}^t} \sum_{(i,j)\in\mathcal{E}} \mathbf{S}_{i,j} \tilde{\mathbf{G}}_{i,j} - \lambda_t \sum_{(i,j)\in\mathcal{E}} \mathbf{S}_{i,j} + \frac{\gamma}{2} \left\| \mathbf{S} - \mathbf{S}^{t-1} \right\|_2^2, \\ &s.t. \quad \mathbf{S}_{i,j} \in [0, 1], \quad \lambda_t = \lambda_0/(T - t + 1), \end{aligned} \tag{15}$$

where λ_t is a variable that controls the number of selected edges and γ is a hyperparameter to control the proximal term. Here, λ_0 represents the base value of λ_t, T denotes the total number of epochs, and t represents the current epoch. As λ_t increases, the number of selected edges also increases, allowing the model to progressively learn additional edges in the process until all edges are selected.

4.3.2 Training Objective

We use $\mathbf{A}^t = \mathbf{S}^t \odot \mathbf{A}$ to denote the learning adjacency matrix at epoch t, we formulate the joint optimization of GNAS and graph structure learning at epoch t as follows:

$$\begin{aligned} &\min_{\mathcal{A},\mathbf{G},\mathbf{W}} \mathcal{L}_{\text{val}}(\mathbf{W}, \mathcal{A}, \mathbf{G}, \mathbf{A}^t) \\ s.t. \quad &\mathcal{A}^* = \arg\min_{\mathcal{A}} \mathcal{L}_{\text{val}}(\mathbf{W}^*, \mathcal{A}, \mathbf{G}^*, \mathbf{A}^t), \\ &\mathbf{G}^* = \arg\min_{\mathbf{G}} \mathcal{L}_s(\mathbf{W}^*, \mathcal{A}, \mathbf{G}, \mathbf{A}^t), \\ &\mathbf{W}^* = \arg\min_{\mathbf{W}} \mathbb{E}_{\mathcal{A}\in\Gamma(\mathcal{A})} \mathcal{L}_{\text{train}}(\mathbf{W}, \mathcal{A}, \mathbf{G}, \mathbf{A}^t) \\ &\quad + \frac{\gamma}{2} \left\| \mathbf{W} - \mathbf{W}^{t-1} \right\|_2^2. \end{aligned} \tag{16}$$

In this optimization, there are three categories of learnable parameters $\mathbf{W}, \mathcal{A}, \mathbf{G}$ that need to be optimized. However, optimizing all of these parameters simultaneously is challenging. Therefore, we apply gradient descent methods to update these parameters alternatively. During the training procedure, we set the first few epochs as a warming-up period, during which only $\mathbf{W}$ is updated. This is because we observe that $\mathbf{W}$ is unstable and may lead to incorrect hidden features at the beginning of training. Therefore, we update only $\mathbf{W}$ during the warming-up period. When the warming period finishes, we use the node-aware approximate nearest neighbor search algorithm to search for potential edges in the graph. Then we select edges to jointly optimize the graph structure and neural architecture. The overall training procedure is described in Algorithm 2, and we prove the convergence of the proposed algorithm.

Proof 4.1.

We first explain that our method satisfies the conditions of a *Proximal Alternating Minimization*. Following this, we introduce a Lemma 2 from [63] to demonstrate the convergence of Algorithm 2.

Property 2. *Proximal Alternating Minimization* is an optimization technique that combines alternating minimization with proximal methods. It works by dividing variables into blocks, alternating between optimizing each block while using a proximal term to control the update magnitude, which prevents numerical instability. A Proximal Alternating Minimization procedure is defined in Algorithm 3.

Algorithm 3 Proximal Alternating Minimization

1: Initialize $(\boldsymbol{x}_0, \boldsymbol{y}_0)$
2: **for** $k = 1, 2, \ldots$ **do**
3: $\quad \boldsymbol{y}_k = \arg\min_{\boldsymbol{y}} f(\boldsymbol{x}_{k-1}, \boldsymbol{y}) + \frac{\gamma}{2} \left\| \boldsymbol{y} - \boldsymbol{y}_{k-1} \right\|_2^2$
4: $\quad \boldsymbol{x}_k = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{y}_k) + \frac{\gamma}{2} \left\| \boldsymbol{x} - \boldsymbol{x}_{k-1} \right\|_2^2$
5: **end for**

Lemma 2. Suppose the sequence $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ generated by Algorithm 3 is bounded and f satisfies the following assumptions. Choose a sufficiently large γ in Algorithm 3. Then Algorithm 3 with random initialization almost surely converges to a second-order stationary point of f.

1. f satisfies the Kurdyka-Łojasiewicz (KL) property and ∇f is Lipschitz continuous on any bounded subset of domain $\mathbb{R}^n \times \mathbb{R}^m$.

2. Any globally smooth function f with $\left\| \nabla^2 f(\boldsymbol{x}, \boldsymbol{y}) \right\| \leq L_f$.

We begin by demonstrating that our algorithm (Algorithm 2) satisfies the conditions of a *Proximal Alternating Minimization*. The objective of our algorithm is to optimize three variables, $\mathbf{W}, \mathcal{A}$, and $\mathbf{G}$, which we treat as a combined block. During the training procedure, we also optimize an additional variable, $\mathbf{S}$, to control the selection of edges. Consequently, our algorithm can be viewed as a *Proximal Alternating Minimization* with two variable groups: $\mathbf{W}, \mathbf{G}, \mathcal{A}$ and $\mathbf{S}$. We optimize the former group in Eq (16), and, given that $\mathbf{A}^t = \mathbf{S}^t \odot \mathbf{A}$, we can rewrite Eq (16) as follows:

$$\mathbf{W}^t, \mathbf{G}^t, \mathcal{A}^t = \arg\min_{\mathbf{W}, \mathbf{G}, \mathcal{A}} \mathcal{L}(\mathbf{W}, \mathbf{G}, \mathcal{A}, \mathbf{S}^t) + \frac{\gamma}{2} \left\| \mathbf{W} - \mathbf{W}^{t-1} \right\|_2^2. \tag{17}$$

We optimize the latter variable group using Eq (15), we can regard $\beta \sum_{(i,j)\in\mathcal{E}} \mathbf{S}_{i,j} \tilde{\mathbf{G}}_{i,j} - \lambda_t \sum_{(i,j)\in\mathcal{E}} \mathbf{S}_{i,j}$ as a function $\mathcal{L}_g(\mathbf{W}^t, \mathbf{G}^t, \mathcal{A}^t, \mathbf{S})$, which can be expressed as follows:

$$\mathbf{S}^t = \arg\min_{\mathbf{S}} \mathcal{L}_g(\mathbf{W}^t, \mathbf{G}^t, \mathcal{A}^t, \mathbf{S}) + \frac{\gamma}{2} \left\| \mathbf{S} - \mathbf{S}^{t-1} \right\|_2^2. \tag{18}$$

Thus, Algorithm 2 constitutes a *Proximal Alternating Minimization*. Next, we demonstrate that our algorithm satisfies the assumptions outlined in Lemma 2.

Since the functions $\mathcal{L}$ and $\mathcal{L}_g$ are coercive and we seek their minimum values, the domain $(\mathbf{W}^t, \mathbf{G}^t, \mathcal{A}^t, \mathbf{S}^t)$ is bounded to prevent these functions from approaching infinite values. Additionally, $\mathcal{L}$ and $\mathcal{L}_g$ satisfy the Kurdyka–Łojasiewicz (KL) property, so our algorithm satisfies assumption 1. Given that the sequence $(\mathbf{W}^t, \mathbf{G}^t, \mathcal{A}^t, \mathbf{S}^t)$ is bounded and the second derivatives of $\mathcal{L}$ and $\mathcal{L}_g$ are continuous, these functions are therefore bounded, *i.e.*, $\max\left\{ \left\| \nabla^2_{\mathbf{W}} \mathcal{L}(\mathbf{W}, \mathbf{G}, \mathcal{A}, \mathbf{S}^t) \right\|, \left\| \nabla^2_{\mathbf{S}} \mathcal{L}_g(\mathbf{W}^t, \mathbf{G}^t, \mathcal{A}^t, \mathbf{S}) \right\| \right\} \leq p$, where p is a constant. Consequently, our algorithm satisfies assumption 2. As a result, our algorithm satisfies both assumptions, and for any $\gamma > p$, our algorithm with random initialization almost surely converges to a second-order stationary point.

Unlike DARTS for image classification tasks, which uses a small proxy model during the search phase and retrains a larger network during the evaluation phase to save memory usage, the number of layers in the super network of Curriculum-GraphLLM is the same as that of a complete GNN. Therefore, we can directly use the super network for evaluation. In fact, we find that the super network performs well in the node classification task, even outperforming the retrained architecture in most cases. This phenomenon may be due to the regularization effect of weight sharing, which can be considered as a type of ensemble model [64]. Other ensemble models, such as dropout [65] and multiple heads [66], have also demonstrated success in deep neural networks. In Curriculum-GraphLLM, these models can be considered an ensemble of different sub-models, which learn different aspects of the data and provide different perspectives on the prediction. These sub-models impact and constrain each other during the training phase, which is a form of regularization that helps reduce over-fitting and improves performance.

4.4 Semantic Denoising: Unified Co-Training for Text-Attributed Graphs

Text-attributed graphs serve as powerful tools by augmenting structural graphs with semantic content, allowing for more advanced and insightful graph modeling. In domains like citation networks [67] and web links [68], this synergy between text and structure helps reveal deeper connections and patterns.

In TAGs, the interplay between textual content and graph structures is crucial for accurate predictions. However, existing GNAS methods have largely overlooked the role of text attributes in optimizing architectures. Our experiments show that enhanced textual representations can lead to more effective architectures, thereby improving text understanding capabilities. Moreover, structure learning during architecture search can synergistically interact with text modeling, enabling better capture of relationships within graph data. This mutual reinforcement forms a cohesive co-training framework introduced in this section that allows these components to jointly optimize one another, leading to superior performance on TAG-related tasks.

4.4.1 Text Attribute Modeling

Text-attributed graphs (TAG) are formulated as $\mathcal{G}_s = (\mathcal{V}, \mathcal{A}, \{\mathbf{s}_n\}_{n\in\mathcal{V}})$, where $\mathbf{s}_n$ is described as the textual information associated with node $\mathbf{n}$. Language models excel at processing textual information in TAGs due to their ability to capture rich contextual semantics and handle diverse language patterns effectively.

Language models employ text encoders, often leveraging transformer architectures, to project text attribute $\mathbf{s}_n$ into latent vector representations $\mathbf{h}_n$. For node classification tasks, the label distribution $\mathbf{y}_n$ for node n can then be obtained by applying an MLP with a softmax layer to $\mathbf{h}_n$. Such a process is structured as follows:

$$\hat{\mathbf{y}}_n = \mathrm{softmax}\left(\mathrm{MLP}(\mathbf{h}_n)\right);\ \mathbf{h}_n = \mathrm{TextEnc}(\mathbf{s}_n). \tag{19}$$

In our research, we emphasize the synergistic interplay among text attribute modeling, neural architecture search, and graph structure learning within the framework of TAGs.

4.4.2 Unified Co-Training Framework for TAGs

For the initial stage in our framework, we freeze the language model's training process for text representation and focus solely on searching and optimizing the graph neural

architecture and structure. This approach is motivated by the need to first establish a solid foundation for those GNN components, denoted as parameters ϕ, while freezing the LM during this phase allows the model to avoid unnecessary interference from text-based features while tuning the underlying structure.

In the next phase, we stabilize the graph neural network architectures, directing the attention towards improving the language model, denoted as parameters θ for more precise and nuanced text modeling. Due to the existence of unobserved nodes lacking ground truth labels, we apply the principles of variational inference. Specifically, we utilize the inferential output of the language model as an approximation to tackle the intractable distributions of unseen data. Subsequently, we minimize the Kullback-Leibler (KL) divergence between the variational distribution and the annotation outcomes produced by the neural architecture. This distribution is embodied through the logits $\mathbf{z}_U^{LM_\theta}$ and $\mathbf{z}_U^{GNN_\phi}$, which act as the latent parameters for the probabilistic model, where U is the set of unlabeled nodes. Correspondingly, let L represent the set of labeled nodes, the training objective can be formulated as:

$$\begin{aligned}\theta^t = \arg\min_\theta \frac{1}{|U|} \mathrm{KL}\left(\mathbf{z}_U^{\mathrm{LM}_{\theta^t}} || \mathbf{z}_U^{\mathrm{GNN}_{\phi^{t-1}}}\right) \\ + \frac{1}{|L|}\sum_{n\in L} -\mathbf{y}_n \log\left(\hat{\mathbf{y}}_n^{\theta^t}\right).\end{aligned} \tag{20}$$

After this phase, the language model benefits from the distributional knowledge transmitted through the GNN architecture, enabling it to capture deeper, contextually enriched text representations.

Once the language model has been fine-tuned, we proceed to utilize the simulated variational distribution of the LM to generate pseudo-labels for the subsequent step, which involves the optimization of graph neural architectures and their associated structures. Specifically, we employ the learned parameters θ^t to infer pseudo labels $\hat{\mathbf{y}}_n^{\theta^t}$ for each unlabeled nodes $n \in U$. Thus, the knowledge encoded by LMs can be distilled into the structural and weight adjustments of GNNs. With both annotated and ground truth labels obtained, the training objective in this step can be written as:

$$\begin{aligned}\phi^t = \arg\min_\phi \frac{1}{|U|}\sum_{n\in U} -\hat{\mathbf{y}}_n^{\theta^{t-1}} \log\left(\hat{\mathbf{y}}_n^{\phi^t}\right) \\ + \frac{1}{|L|}\sum_{n\in L} -\mathbf{y}_n \log\left(\hat{\mathbf{y}}_n^{\phi^t}\right),\end{aligned} \tag{21}$$

where λ_ϕ is a hyperparameter similar to λ_θ. For the second term, the observed labels are used as supervision, while the former term acts as a knowledge injection process on those unobserved nodes. Excluding the initialization phase, the two subsequent stages are performed alternately until the model reaches convergence. The overall algorithm is shown in Algorithm 4.

Algorithm 4 Unified Co-Training For TAGs

Require: Graph feature matrix $\mathbf{X}$, the adjacent matrix $\mathbf{A}$, textual attributes $\{\mathbf{s}_n\}$, epoch intervals for dual training n_i, the number of warming up epoches n_w.

```
Initialize the language model LM with parameters θ
Initialize the parameters φ outlined in Algorithm 2
while not converge do
  if epoch <= n_w then
    Freeze θ, optimize φ by Algorithm 2
    Add potential edges following Algorithm 2
  else if epoch > n_w then
    if epoch reaches k n_i (k ∈ ℕ) then
      Calculate z_U^{LM_θ} with θ, z_U^{GNN_φ} with φ
      Freeze φ, optimize θ with Eq (20)
    else
      Annotate pseudo-labels ŷ_n^θ with θ
      Freeze θ, optimize φ with Eq (21) and Algorithm 2
    end if
  end if
end while
Return f_{θ,φ}(X, {s_n})
```

4.5 Complexity Analysis

4.5.1 Complexity of Adding Potential Edges

We first analyze the time complexity of hierarchical small-world graphs. In this process, a node must traverse from the top-layer graph to the bottom-layer graph. The search at each layer always terminates before reaching a node that belongs to the upper layer; otherwise, the search process in the upper layer would have reached a new node. Since a node has a p probability of appearing in the upper graph, the expected number of steps in a layer is $\frac{1}{1-p}$. For each step, it needs to traverse all edges connected to the current node, which costs an amount of time equal to the average degree $\frac{|\mathcal{E}|}{|\mathcal{V}|}$ in expectation. Since there are $O(\log|\mathcal{V}|)$ layers, the time complexity of inserting one node is $O(\frac{|\mathcal{E}|\log|\mathcal{V}|}{|\mathcal{V}|})$, thus the overall complexity of inserting all nodes is $O(|\mathcal{E}|\log|\mathcal{V}|)$. The process of our proposed node-aware approximate nearest neighbor search is similar, which also has a time complexity of $O(|\mathcal{E}|\log|\mathcal{V}|)$.

4.5.2 Complexity in Structure Learning

For the structure learning part, as we only allow each node to have at most twice the original degree of edges in the small world graph, there are at most $2|\mathcal{E}|$ potential edges. Combined with the original edges, there are at most $3|\mathcal{E}|$ edges. In the structure learning process, we consider optimizing the parameter matrix for these edges. Therefore, the time complexity of structure learning is $O(|\mathcal{E}|)$.

4.5.3 Complexity in Automatic Edge Selection

For the automatic edge selection process, it is necessary to employ a mask whose length corresponds to the number of edges, as there are at most $3|\mathcal{E}|$ edges. This mask $\mathbf{S}$ is applied to all edges through the dot product. Consequently, the time complexity of the automatic edge selection is $O(|\mathcal{E}|)$.

4.5.4 Memory Cost

The expectation of the maximum number of edges in the hierarchical small world graphs is $\frac{2|\mathcal{E}|}{1-p}$. In addition, the memory cost in structure learning is $3|\mathcal{E}|$. Other parts of Curriculum-GraphLLM such as the supernet also have a memory cost of $O(|\mathcal{E}|)$. Thus, the overall memory cost is $O(|\mathcal{E}|)$.

4.5.5 Overall Complexity

In summary, our proposed model has $O(|\mathcal{E}|\log|\mathcal{V}|) = O(\bar{d}|\mathcal{V}|\log|\mathcal{V}|)$ time complexity and $O(|\mathcal{E}|)$ memory cost, where $\bar{d}$ is the average degree. Since most real-world graphs

are sparse [69], the average degree of nodes can be considered as a small constant, *i.e.*, $\bar{d} \ll |\mathcal{V}|$. As a result, the time complexity of our method is comparable to previous GNAS methods, which usually has a time complexity $O(\bar{d}|\mathcal{V}|)$, and is much more efficient than graph structure learning approaches which cost $O(|\mathcal{V}|^2)$.

5 Experiments

In this section, we report experimental results on real-world graphs to validate the effectiveness of our proposed model.

5.1 Experimental Setting

We evaluate our model on six widely used graph benchmark datasets: Cora, CiteSeer, PubMed [70], Coauthor-CS, Coauthor-Physics, and Amazon-Photo [71]. For Cora, CiteSeer, and PubMed, we apply the full-supervised training setup used in [72]. For Coauthor and Amazon datasets, we randomly split the training/validation/testing set using a ratio of 60%:20%:20%.

We compare our method with various baselines, including handcrafted GNNs: GCN [5], GAT [6], and ARMA [73], representative NAS algorithms: DARTS [15], GDAS [74], ASAP [75], XNAS [76], GraphNAS [10], GAUSS [77] and D2GNAS [78], graph structure learning methods: GRCN [79], NodeFormer [80] and HES-GSL [81] and graph curriculum learning methods CLNode [47], RCL [49] and TSS [82]. The hyperparameters of baselines are kept the same as in their original implementation. For our model, we set the number of layers as 2. The candidate operations contain *GCN* [5], *GAT* [6], *GrpahSAGE* [83], *ARMA* [73] and *linear*. We also compare two versions of our method: the full model and GASSO, *i.e.*, only removing noisy edges but without adding potential edges and curriculum strategies.

For TAG node classification tasks, we adopt the datasets and configurations in line with GLBench [84] and perform experiments on Cora, Citeseer, Pubmed [85], WikiCS [86] and Instagram [87]. All dataset splits strictly follow the default settings given by the benchmark's corresponding files. Our method's performance was evaluated against that of the majority of baseline approaches on TAG benchmarks, including handcrafted GNNs GCN [5], GAT [6], and GraphSAGE [83], NAS algorithms DARTS [15], GAUSS [77], and D2GNAS [78], and all aforementioned graph structure learning and curriculum learning methods. Additionally, several pretrained language models (PLMs) with varying parameter sizes: Sentence-BERT (22M) [88], BERT (110M) [89], and RoBERTa (355M) [90], and several state-of-the-art techniques that fuse LLM and GNN: GIANT [91], OFA [86], TAPE [57], ENGINE [92], GraphText [93], GraphAdapter [87], LLaGA [94] and GLEM [95] are likewise taken into account in the comparison. For the baseline results of LLM-based methods, we directly leverage those reported in GLBench [84]. Although our approach employs Sentence-BERT [88] as the LM backbone, it can be easily extended to incorporate larger language models such as LLaMA [96]. Therefore, for a fair comparison, we also include methods utilizing LLaMA in their implementation with a greater number of parameters in our evaluation.

5.2 Performance Evaluation

We summarize the results of node classification on graphs without semantic attributes in Table 2, and on TAGs in Table 3. All results in the table are averaged over 5 runs with random parameter initialization.

The results indicate that our proposed model, Curriculum-GraphLLM, can effectively leverage the information contained in the original graph, as well as the semantic insights captured through text modeling, resulting in improved node label predictions. In particular, our model achieves consistently competitive or superior average performance compared to the baselines, demonstrating its effectiveness in node classification tasks under both graph settings. It is worth noting that the absolute improvements on several clean graph benchmarks are moderate. This is expected because these widely used datasets have become highly saturated, and many recent baselines already achieve strong performance under clean graph settings.

5.3 Denoising Analysis

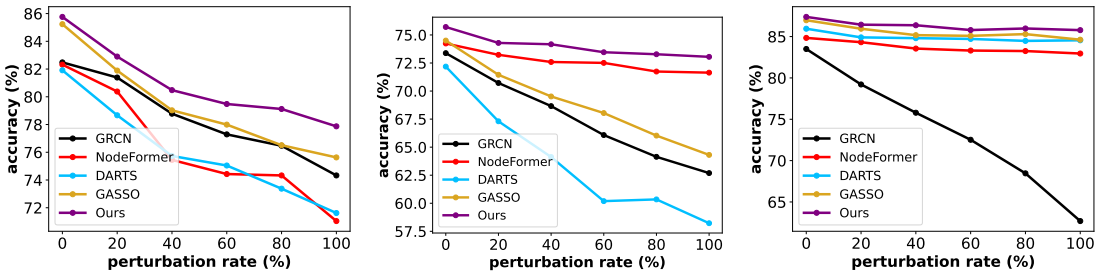
In this section, we evaluate the denoising ability of our proposed model. We follow the experimental settings used in previous works [29], [97], [98], where only the largest connected component of the graph is considered, and the training/validation/testing sets are split at $10\%/10\%/80\%$. We use two approaches to add noise to the graph structure: **adding** and **flipping**. For the **adding** setting, we manually add random edges to the datasets with different perturbation rates, ranging from 0% to 100% with a step of 20%. For the **flipping** setting, we manually flip the existence or non-existence of edges between pairs of random nodes, using perturbation rates from 0% to 100% as well. We compare our proposed methods with baselines of graph structure learning GRCN and Nodeformer along with NAS algorithms DARTS and GASSO. We use our model to examine the effect of denoising, as well as mining potential connections. The results are shown in Figure 3 and Figure 4.

We observe that DARTS has an unstable denoising ability. It performs very well on PubMed, where the accuracy almost remains the same as on the original graph even when there is a 100% perturbation rate of edges. However, it performs poorly in the other two cases (Cora and CiteSeer). Structure learning baselines exhibit such instability as well. Meanwhile, although GRCN shows significant shortcomings under the two denoising configurations on PubMed, GSL approaches demonstrate a clear advantage over DARTS across other experimental conditions. Our proposed model consistently outperforms the baselines in most cases of the two settings and all three datasets. Our model leads a margin of around 15% in CiteSeer and 6% in Cora compared to DARTS at a 100% perturbation rate in the **adding** setting. Moreover, the margins increase to 16% in CiteSeer and 12% in Cora at a 100% perturbation rate in the **flipping** setting. Those figures shrink by an average of 50% when GSL methods are applied in corresponding settings. However, a substantial gap still exists between them and our model, indicating that our approach greatly benefits the prediction task in such highly noisy graphs. Additionally, our model outperforms GASSO in all cases, verifying the usefulness of adding potential edges and graph curriculum strategy.

TABLE 2: The performance of node classification on six benchmark graph datasets. The average performance and standard deviations over 5 runs are reported. The best results are in **bold**, while the second-best methods are underlined for emphasis.

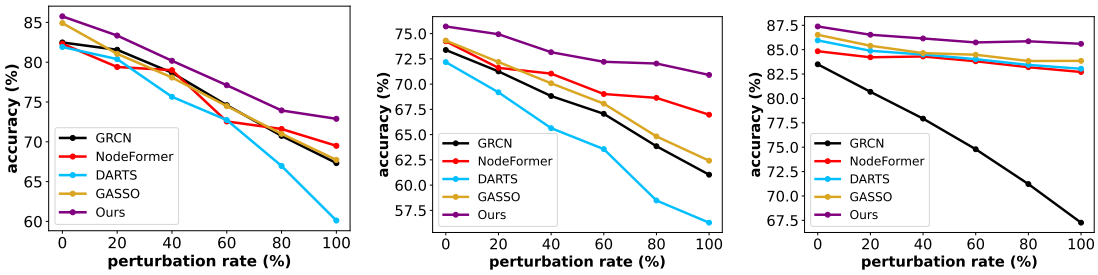
Dataset	Cora	CiteSeer	PubMed	CS	Physics	Photo
GCN (2017)	87.38 ± 0.15	79.02 ± 0.07	88.24 ± 0.20	93.36 ± 0.37	95.94 ± 0.16	94.02 ± 0.34
GAT (2018)	87.26 ± 0.08	77.82 ± 0.11	86.83 ± 0.11	92.42 ± 0.55	95.89 ± 0.23	94.02 ± 0.30
ARMA (2022)	86.06 ± 0.05	76.50 ± 0.00	88.70 ± 0.24	94.18 ± 0.21	96.57 ± 0.10	94.46 ± 0.50
DARTS (2019)	86.18 ± 0.36	74.96 ± 0.10	88.38 ± 0.18	93.57 ± 0.76	96.31 ± 0.37	89.73 ± 5.15
GASSO (2021)	87.63 ± 0.29	79.61 ± 0.32	90.52 ± 0.24	94.45 ± 0.36	96.84 ± 0.21	93.74 ± 0.46
GDAS (2019)	85.48 ± 0.30	74.20 ± 0.11	89.50 ± 0.14	93.74 ± 0.34	96.56 ± 0.35	94.39 ± 0.59
ASAP (2020)	85.21 ± 0.13	75.14 ± 0.09	88.65 ± 0.10	92.40 ± 0.29	95.98 ± 0.21	93.62 ± 0.32
XNAS (2019)	86.80 ± 0.14	76.33 ± 0.09	88.61 ± 0.25	92.55 ± 1.24	95.78 ± 0.63	90.60 ± 5.07
GraphNAS‡ (2020)	86.83 ± 0.56	79.05 ± 0.28	89.99 ± 0.43	93.90 ± 0.12	96.31 ± 0.07	93.13 ± 0.28
GAUSS (2022)	87.10 ± 0.25	78.56 ± 0.56	88.64 ± 0.33	93.78 ± 0.11	96.44 ± 0.05	94.13 ± 0.07
D2GNAS (2024)	86.65 ± 0.35	77.47 ± 0.50	89.12 ± 0.27	95.12 ± 0.08	96.60 ± 0.04	94.64 ± 0.20
GRCN (2020)	86.30 ± 0.33	75.25 ± 0.41	84.20 ± 0.19	94.69 ± 0.01	95.40 ± 0.06	93.74 ± 0.12
NodeFormer (2022)	86.96 ± 0.09	79.62 ± 0.04	90.20 ± 0.22	95.16 ± 0.04	95.99 ± 0.03	93.45 ± 0.30
HES-GSL (2024)	77.78 ± 1.11	76.24 ± 0.93	88.24 ± 0.44	90.49 ± 0.45	94.75 ± 0.25	85.67 ± 1.09
CLNode (2023)	86.60 ± 0.64	78.99 ± 0.57	89.50 ± 0.28	94.13 ± 0.34	96.87 ± 0.45	93.90 ± 0.42
RCL (2023)	87.15 ± 0.44	79.79 ± 0.55	90.14 ± 0.43	95.06 ± 0.12	96.86 ± 0.16	94.46 ± 0.18
TSS (2024)	86.24 ± 0.47	77.14 ± 0.72	86.10 ± 0.33	88.55 ± 0.10	94.83 ± 0.50	80.30 ± 1.24
Curriculum-GraphLLM (Ours)	$\mathbf{88.00 \pm 0.29}$	$\mathbf{79.87 \pm 0.09}$	$\mathbf{90.87 \pm 0.38}$	$\mathbf{95.47 \pm 0.19}$	$\mathbf{96.98 \pm 0.06}$	$\mathbf{94.74 \pm 0.16}$

‡ We rerun GraphNAS because their original code leaks the test data set.



(a) Cora (b) CiteSeer (c) PubMed

Fig. 3: The performance under **adding** perturbation settings.



(a) Cora (b) CiteSeer (c) PubMed

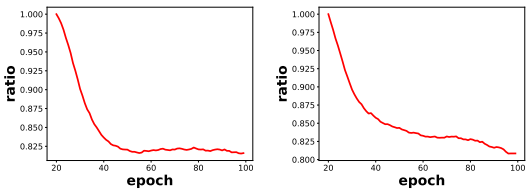
Fig. 4: The performance under **flipping** perturbation settings.

To better demonstrate the effectiveness of our denoising process in the **adding** setting, we show the ratio in weights of noisy edges compared to original edges in Figure 5. We choose the case of 100% perturbation rate in Cora and CiteSeer. The first 20 epochs are the warm-up stage and thus are not shown in the figure. We can observe that the weight ratio continues to decrease and tends to stabilize at around 82.5% in both cases, indicating that we can exclude 17.5% of the noise effect. All the above results show that our model is very effective in reducing noise that can interfere with NAS methods.

We also show the probability ratio that a flipped edge should be added back compared to a non-existent edge under the **flipping** setting in Figure 6 to demonstrate the effectiveness of our edge addition process. We find that in all three datasets, the flipped edges have a much higher probability of being recovered by our approach than other non-existent edges. For example, in PubMed, the deleted edges have an 80 times higher probability of being recovered by our method than non-existent edges. This ability makes our model more stable in graphs with a lot of missing edges.

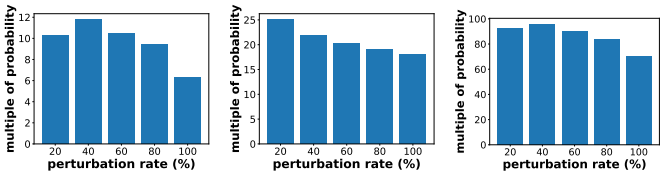
TABLE 3: The performance of node classification on five TAG datasets. The average performance and standard deviations over 3 runs are reported. The highest results are highlighted in **bold**, while the second-best methods are underlined for emphasis. Rank means the average rank among all datasets. OOM means out of memory.

Dataset	Cora	CiteSeer	PubMed	Instagram	WikiCS	Rank
GCN (2017)	76.72 ± 0.94	67.85 ± 0.39	76.49 ± 0.39	66.24 ± 0.50	79.96 ± 0.06	10.60
GAT (2018)	77.97 ± 0.51	67.58 ± 0.47	76.54 ± 0.41	65.20 ± 0.08	79.80 ± 0.29	11.60
GraphSAGE (2017)	79.71 ± 0.53	67.23 ± 0.87	76.38 ± 0.69	66.64 ± 0.10	79.00 ± 0.22	10.60
DARTS (2019)	77.14 ± 0.57	65.50 ± 3.53	77.38 ± 1.75	65.50 ± 1.40	76.48 ± 0.41	14.20
GASSO (2021)	80.50 ± 1.11	68.07 ± 0.53	78.99 ± 2.47	65.00 ± 1.73	78.17 ± 0.29	9.00
GAUSS (2022)	80.32 ± 1.22	66.65 ± 1.19	78.66 ± 1.63	63.84 ± 0.22	79.77 ± 0.27	10.50
D2GNAS (2024)	78.35 ± 1.92	63.26 ± 4.12	77.86 ± 2.37	66.36 ± 0.93	79.27 ± 0.07	12.10
GRCN (2020)	82.29 ± 1.05	71.49 ± 1.79	78.58 ± 1.16	66.66 ± 0.07	77.52 ± 0.04	7.00
NodeFormer (2022)	78.58 ± 2.39	67.63 ± 0.68	76.20 ± 1.27	66.53 ± 0.50	73.48 ± 0.95	12.80
HES-GSL (2023)	73.34 ± 0.43	64.94 ± 3.26	77.68 ± 0.81	63.33 ± 0.32	61.21 ± 0.57	18.40
CLNode (2023)	82.09 ± 0.89	66.85 ± 2.07	78.59 ± 2.01	63.76 ± 0.31	78.95 ± 0.75	10.60
RCL (2023)	78.35 ± 1.51	64.51 ± 3.26	78.96 ± 0.95	65.62 ± 0.48	78.28 ± 0.38	12.10
TSS (2024)	81.06 ± 1.03	65.41 ± 2.74	78.89 ± 1.24	63.55 ± 0.01	78.79 ± 0.44	11.60
Sent-BERT (22M) (2019)	55.58 ± 4.94	58.00 ± 1.76	60.09 ± 2.50	64.65 ± 0.36	64.71 ± 3.67	22.00
BERT (110M) (2019)	56.91 ± 2.74	62.09 ± 1.49	57.37 ± 2.66	62.02 ± 0.47	78.07 ± 0.48	21.60
RoBERTa (335M) (2019)	60.27 ± 7.43	67.76 ± 1.99	64.22 ± 0.11	59.26 ± 1.39	80.41 ± 0.28	16.60
GIANT (2022)	79.92 ± 0.22	69.89 ± 0.88	76.65 ± 1.41	64.31 ± 0.96	80.73 ± 0.16	9.20
TAPE-TA (2024)	80.32 ± 1.18	64.95 ± 1.01	75.11 ± 1.29	65.17 ± 0.40	79.70 ± 0.46	13.10
OFA (2024)	73.81 ± 0.86	64.72 ± 0.64	74.55 ± 0.62	60.34 ± 0.15	77.78 ± 0.04	19.40
ENGINE (2024)	81.58 ± 0.65	71.12 ± 0.76	75.94 ± 0.72	67.17 ± 0.53	81.64 ± 0.35	6.20
GraphText (2023)	70.81 ± 1.23	62.56 ± 0.83	75.78 ± 0.81	65.27 ± 0.45	67.08 ± 7.11	19.00
GraphAdapter (2024)	71.82 ± 0.32	68.73 ± 1.09	72.14 ± 0.23	$\mathbf{67.51 \pm 0.20}$	70.40 ± 0.34	14.00
LLaGA (2024)	73.24 ± 0.13	30.84 ± 0.26	43.08 ± 0.12	60.77 ± 0.31	63.07 ± 0.23	23.20
GLEM (2023)	79.19 ± 0.25	71.62 ± 0.60	79.64 ± 3.56	OOM	82.16 ± 0.19	8.40
Curriculum-GraphLLM (Ours)	$\mathbf{83.91 \pm 0.81}$	$\mathbf{73.99 \pm 1.14}$	$\mathbf{81.46 \pm 0.74}$	67.44 ± 0.18	$\mathbf{82.99 \pm 0.06}$	**1.20**



(a) Cora (b) CiteSeer

Fig. 5: The ratio in weights of noisy edges to the original edges in the graph under the adding perturbation setting.



(a) Cora (b) CiteSeer (c) PubMed

Fig. 6: The probability ratio that a flipped edge should be added back compared to a non-existent edge under the flipping perturbation setting.

5.4 Ablation Study

5.4.1 *Non-text-attributed graphs*

On non-text-attributed graphs, our proposed Curriculum-GraphLLM method comprises three crucial components: curriculum learning, graph neural architecture search and graph structure learning. To evaluate the contribution of these components and their cooperative patterns, we conduct ablation studies in this section, where we design several variants of the model classified into three groups.

(1) The first group of variants modifies the graph structure learning model with different regularization of smoothness. Note that none of these variants can add potential edges. Specifically, we consider five variants:

- noCur: we remove the curriculum learning strategy and directly use a joint optimization of structure and architecture.
- noStru: we remove the structure learning part and use our initial $\mathbf{G}$ so that the model becomes equivalent to

TABLE 4: The results of ablation studies.

Dataset	Cora	CiteSeer	PubMed
Curriculum-GraphLLM	$\mathbf{88.00 \pm 0.29}$	$\mathbf{79.87 \pm 0.09}$	$\mathbf{90.87 \pm 0.38}$
GASSO	87.63 ± 0.29	79.61 ± 0.32	90.52 ± 0.24
-noCur	87.67 ± 0.27	79.69 ± 0.35	90.63 ± 0.31
-noStru	87.19 ± 0.24	78.71 ± 0.16	90.30 ± 0.46
-oriSmooth	87.17 ± 0.26	78.62 ± 0.22	90.13 ± 0.33
-labSmooth	87.35 ± 0.19	78.80 ± 0.33	90.26 ± 0.44
-simWeight	87.31 ± 0.21	79.04 ± 0.34	89.12 ± 0.18
-NoNAS	86.44 ± 0.43	77.72 ± 0.33	89.09 ± 0.39
-GCN†	87.22 ± 0.64	79.31 ± 0.35	86.55 ± 0.28
-GAT†	86.55 ± 0.28	78.13 ± 0.68	87.46 ± 0.28
-Stru-NAS	87.14 ± 0.37	78.38 ± 0.24	90.40 ± 0.39
-NAS-Stru	87.37 ± 0.24	78.22 ± 0.42	90.24 ± 0.30

† The number of channels in these models is expanded to ensure that they have a comparable number of parameters to Curriculum-GraphLLM.

DARTS.

- oriSmooth: we replace the hidden feature smoothness with the original feature smoothness used in [29].
- labSmooth: we replace the hidden feature smoothness with one-hot predicted label smoothness similar to [30].
- simWeight: we directly calculate edge weight based on node similarities.

(2) The second group of variants alters the neural architecture search method. We consider three variants:

- noNAS: we do not update the neural architecture, but the super network is still kept with all candidate operations having equal weights.
- GCN: we fix GCN as the architecture.
- GAT: we fix GAT as the architecture.

(3) The third group of variants modifies the optimization scheme to validate the cooperation of graph structure learning and neural architecture search. We consider two variants:

- Stru-NAS: we split the training of neural architecture and graph structure and only train the graph structure in the first half of epochs and train the neural architecture in the second half. To ensure a fair comparison, the number of training epochs for this variant is doubled, and the warming-up procedure is kept unchanged.
- NAS-Stru: the same procedure as Stru-NAS, but the order of training neural architecture and graph structure is exchanged.

We evaluate all variants on Cora, CiteSeer, and PubMed datasets, and the results are presented in Table 4. Overall, our proposed Curriculum-GraphLLM model outperforms all variants in all three datasets, demonstrating that optimizing both graph structure and neural architecture search contributes to our model.

For the first group of variants, we find that no-Cur drops significantly, indicating that a curriculum strategy greatly benefits the joint optimization of structure and architecture. We observe that NoStru demonstrates performance comparable to that of GCN and GAT, which highlights the significant potential of ensembling within the super network. Nevertheless, there remains a performance gap between these models and ours, underscoring the necessity of graph structure learning. Interestingly, the performance of oriSmooth is not as good as noStru, suggesting that using original feature smoothness to constrain edges may cause more severe overfitting. The performance of labSmooth indicates that one-hot predicted label smoothness is somewhat useful, but neither the original feature nor the one-hot predicted label is better than the hidden feature for graph structure learning. SimWeight is a radical method that makes the edge weights different at the early stage of training.

For the second group of variants, we observe that performance significantly drops in NoNAS, demonstrating that neural architecture training is of significant importance. GCN and GAT variants behave differently in the three datasets, indicating that these datasets have different properties.

For the third group of variants, we notice that Stru-NAS can be regarded as noNAS followed by neural architecture search, and NAS-Stru can be regarded as noStru followed by graph structure learning. The margin of Stru-NAS over noNAS and the margin of NAS-Stru over noStru demonstrate that adding neural architecture search after graph structure learning achieves better performance, while adding graph structure learning after neural architecture search causes unstable influence. Nevertheless, neither Stru-NAS nor NAS-Stru outperforms Curriculum-GraphLLM, demonstrating that the cooperative training in our proposed method plays a key role in optimizing the model.

TABLE 5: The results of ablation studies on TAGs.

Dataset	Cora	CiteSeer	PubMed
Curriculum-GraphLLM (Ours)	**83.91**	**73.99**	**81.46**
GASSO	80.50	68.07	78.99
-noCoTrain	81.87	71.12	80.66
-noCur	81.58	70.34	80.56
-noAdd	81.82	69.94	80.57
-noStru	81.43	69.88	80.02
-oriSmooth	81.28	69.68	80.03
-labSmooth	81.24	69.52	79.22
-simWeight	81.62	69.91	78.91

5.4.2 *Text-attributed graphs*

On text-attributed graphs, we primarily analyze the contribution of the unified co-training module. We conducted experiments by eliminating the co-training module (-noCoTrain in the table), relying solely on the raw text representations from graphs for training without any fine-tuning applied to the language model's text encoder. The experimental setup for the datasets strictly adheres to the benchmark protocols. In addition, we conducted similar ablation studies in alignment with the previous section's paradigm. Results are portrayed in Table 5. Note that the corresponding items in the table mirror those configured in the non-TAG framework, while the -noAdd item refers to the omission of potential edge addition.

The findings demonstrate that unified co-training not only enhances the modeling of textual information in text-attributed graphs but also strengthens the interplay between the language model and GNN architecture and structure components. This dual mechanism improves representation learning by integrating both node-level features and contextual information. Furthermore, the ablation experiments conducted in accordance with the aforementioned section under the TAG setting provide additional evidence of the crucial role that these components play in handling diverse graph patterns.

6 Discussion and Conclusion

6.1 Relationship with Graph Foundation Models

Foundation models, including graph Transformers and graph foundation models, have shown strong potential in graph representation learning [44], [57], [95], [99]. Nevertheless, Curriculum-GraphLLM is not intended to replace over-parameterized foundation models in data-rich and clean settings. Instead, it provides a complementary task-adaptive framework for noisy, label-scarce, domain-specific, and resource-constrained graph scenarios, where task-specific architecture and topology optimization remain important. In particular, large model capacity does not automatically

eliminate structural noise caused by missing or redundant edges, while our dynamic topology updating and curriculum-based edge selection explicitly refine graph structures during architecture search. Therefore, Curriculum-GraphLLM can also complement graph foundation models as a topology refinement module, lightweight downstream adapter, or student architecture search component.

6.2 Conclusion

In this paper, we explore graph neural architecture search in the presence of *structural and semantic noise*. Specifically, we conduct a systematic investigation into how graph architecture search can select optimal architectures through a measurement study. Our findings indicate that differentiable graph architecture search is capable of selecting suitable operations by leveraging the informativeness in the graph. However, it may be susceptible to noise present in node features and graph structures. Based on these findings, we propose our Curriculum-GraphLLM model, which jointly optimizes graph architecture and structure with a curriculum strategy, as well as introducing a unified co-training module to enhance the understanding of the text semantics. As such, the proposed Curriculum-GraphLLM is able to handle *structural and semantic noise* simultaneously. Extensive experiments conducted on real-world graph benchmark datasets demonstrate that our method not only enhances graph learning performance but also improves its denoising capability.

REFERENCES

[1] L. Liao, X. He, H. Zhang, and T.-S. Chua, "Attributed social network embedding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 12, pp. 2257–2270, 2018.

[2] Y.-L. Hsu, Y.-C. Tsai, and C.-T. Li, "Fingat: Financial graph attention networks for recommending top-k k profitable stocks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 1, pp. 469–481, 2021.

[3] J. Sun, J. Zhang, Q. Li, X. Yi, Y. Liang, and Y. Zheng, "Predicting citywide crowd flows in irregular regions using multi-view graph convolutional networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 5, pp. 2348–2359, 2020.

[4] T. Fu, C. Xiao, L. M. Glass, and J. Sun, "Moler: incorporate molecule-level reward to enhance deep generative model for molecule optimization," *IEEE transactions on knowledge and data engineering*, vol. 34, no. 11, pp. 5459–5471, 2021.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[6] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[7] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2018.

[8] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 249–270, 2020.

[9] G. Li, G. Qian, I. C. Delgadillo, M. Muller, A. Thabet, and B. Ghanem, "Sgas: Sequential greedy architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1620–1630.

[10] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graph neural architecture search," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, 2020, pp. 1403–1409.

[11] Y. Gao, P. Zhang, H. Yang, C. Zhou, Z. Tian, Y. Hu, Z. Li, and J. Zhou, "Graphnas++: Distributed architecture search for graph neural networks," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[12] Y. Gao, P. Zhang, C. Zhou, H. Yang, Z. Li, Y. Hu, and S. Y. Philip, "Hgnas++: efficient architecture search for heterogeneous graph neural networks," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[13] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.

[14] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *ICML*, 2018.

[15] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *ICLR*, 2019.

[16] Z. Zhang, X. Wang, and W. Zhu, "Automated machine learning on graphs: A survey," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 2021, survey track.

[17] Y. Qin, X. Wang, Z. Zhang, and W. Zhu, "Graph differentiable architecture search with structure learning," in *NeurIPS*, 2021.

[18] Y. Qin, X. Wang, Z. Zhang, H. Chen, and W. Zhu, "Multi-task graph neural architecture search with task-aware collaboration and curriculum," *NeurIPS*, vol. 36, pp. 24 879–24 891, 2023.

[19] Z. Zhang, X. Wang, Z. Zhang, G. Shen, S. Shen, and W. Zhu, "Unsupervised graph neural architecture search with disentangled self-supervision," *NeurIPS*, vol. 36, pp. 73 175–73 190, 2023.

[20] B. Xie, H. Chang, Z. Zhang, Z. Zhang, S. Wu, X. Wang, Y. Meng, and W. Zhu, "Towards lightweight graph neural network search with curriculum graph sparsification," in *KDD*, 2024, pp. 3563–3573.

[21] P. Li, X. Wang, Z. Zhang, Z. Zhang, F. Shen, J. Wang, Y. Li, and W. Zhu, "Causal-aware graph neural architecture search under distribution shifts," in *KDD*, 2025, pp. 1458–1469.

[22] G. Zhu, Z. Ji, J. Chen, L. Wang, C. Yuan, and Y. Huang, "Sa-gnas: Seed architecture expansion for efficient large-scale graph neural architecture search," *arXiv preprint arXiv:2412.02196*, 2024.

[23] H. Wang, Y. Gao, X. Zheng, P. Zhang, J. Bu, and P. S. Yu, "Graph neural architecture search with large language models," *Sci. China Inf. Sci.*, vol. 68, no. 12, p. 222103, 2025.

[24] H. Chen, X. Wang, Z. Zhang, H. Li, L. Feng, and W. Zhu, "Autogfm: Automated graph foundation model with adaptive architecture customization," in *ICML*, 2025.

[25] K. Zhou, X. Huang, Q. Song, R. Chen, and X. Hu, "Auto-gnn: Neural architecture search of graph neural networks," *Frontiers in big Data*, vol. 5, p. 1029307, 2022.

[26] Y. Gao, P. Zhang, H. Yang, C. Zhou, Y. Hu, Z. Tian, Z. Li, and J. Zhou, "Graphnas++: Distributed architecture search for graph neural networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 6973–6987, 2022.

[27] Y. Ding, Q. Yao, H. Zhao, and T. Zhang, "Diffmg: Differentiable meta graph search for heterogeneous graph neural networks," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 279–288.

[28] X. Zheng, M. Zhang, C. Chen, Q. Zhang, C. Zhou, and S. Pan, "Autoheg: Automated graph neural network on heterophilic graphs," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 611–620.

[29] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *KDD*, 2020, pp. 66–74.

[30] L. Yang, Z. Kang, X. Cao, D. Jin, B. Yang, and Y. Guo, "Topology optimization based graph convolutional network," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019, pp. 4054–4061.

[31] S. Han, Z. Zhou, J. Chen, Z. Hao, S. Zhou, G. Wang, Y. Feng, C. Chen, and C. Wang, "Uncertainty-aware graph structure learning," in *WWW*, 2025, pp. 4863–4874.

[32] Y. Zheng, Z. Zhang, Z. Wang, X. Li, S. Luan, X. Peng, and L. Chen, "Rethinking structure learning for graph neural networks," *arXiv preprint arXiv:2411.07672*, 2024.

[33] Z. Guo, L. Xia, Y. Yu, Y. Wang, K. Lu, Z. Huang, and C. Huang, "Graphedit: Large language models for graph structure learning," *arXiv preprint arXiv:2402.15183*, 2024.

[34] Y. In, K. Yoon, K. Kim, K. Shin, and C. Park, "Self-guided robust graph structure refinement," in *WWW*, 2024, pp. 697–708.

[35] T. Yang, J. Meng, M. Zhou, Y. Yang, Y. Wang, X. Li, and Y. Tong, "You can't ignore either: Unifying structure and feature denoising for robust graph learning," in *CIKM*, 2024, pp. 4178–4182.

[36] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis, "All you need is low (rank): Defending against adversarial attacks on graphs," in *The Thirteenth ACM International Conference on Web Search and Data Mining*, 2020, pp. 169–177.

[37] H. Chen, X. Wang, G. Chen, Y. Meng, H. Li, Y. Yao, Z. Zhang, Z. Zhang, J. Zhou, L. Feng *et al.*, "Adaptive mixture of disentangled

experts for dynamic graph out-of-distribution generalization," in *ICLR*.

[38] S. Tian, X. Wang, Z. Zhang, H. Chen, and W. Zhu, "Out-of-distribution generalized graph anomaly detection with homophily-aware environment mixup," *NeurIPS*, vol. 38, pp. 65 681–65 705, 2026.

[39] X. Wang, H. Li, Z. Zhang, H. Chen, T. Xiao, K. Li, and W. Zhu, "Uncertainty-aware disentangled dynamic graph attention network for out-of-distribution generalization," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2025.

[40] H. Li, X. Wang, Z. Zhang, H. Chen, Z. Zhang, and W. Zhu, "Disentangled graph self-supervised learning for out-of-distribution generalization," in *ICML*, 2024.

[41] Z. Zhang, X. Wang, H. Chen, H. Li, and W. Zhu, "Disentangled dynamic graph attention network for out-of-distribution sequential recommendation," *ACM Trans. Inf. Syst.*, vol. 43, no. 1, pp. 1–42, 2024.

[42] Y. Zhu, W. Xu, J. Zhang, Q. Liu, S. Wu, and L. Wang, "Deep graph structure learning for robust representations: A survey," *arXiv preprint arXiv:2103.03036*, 2021.

[43] X. Wang, M. Zhu, D. Bo, P. Cui, C. Shi, and J. Pei, "Am-gcn: Adaptive multi-channel graph convolutional networks," in *KDD*, 2020, pp. 1243–1253.

[44] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" *NeurIPS*, vol. 34, pp. 28 877–28 888, 2021.

[45] H. Wang, K. Zhou, X. Zhao, J. Wang, and J.-R. Wen, "Curriculum pre-training heterogeneous subgraph transformer for top-n recommendation," *ACM Transactions on Information Systems*, vol. 41, no. 1, pp. 1–28, 2023.

[46] H. Li, X. Wang, and W. Zhu, "Curriculum graph machine learning: A survey," *arXiv preprint arXiv:2302.02926*, 2023.

[47] X. Wei, X. Gong, Y. Zhan, B. Du, Y. Luo, and W. Hu, "Clnode: Curriculum learning for node classification," in *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, 2023, pp. 670–678.

[48] X. Li, Z. Fan, F. Huang, X. Hu, Y. Deng, L. Wang, and X. Zhao, "Graph neural network with curriculum learning for imbalanced node classification," *Neurocomputing*, vol. 574, p. 127229, 2024.

[49] Z. Zhang, J. Wang, and L. Zhao, "Curriculum learning for graph neural networks: Which edges should we learn first," *Advances in neural information processing systems*, vol. 36, pp. 51 113–51 132, 2023.

[50] X. Wang, L. Xiao, Y. Yao, and W. Zhu, "Ood-graphllm: Graph large language model for out-of-distribution generalized drug synergy prediction," in *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, 2026, pp. 12 315–12 326.

[51] H. Chen, X. Wang, Z. Zhang, H. Li, W. Wen, L. Feng, and W. Zhu, "Curriculum gnn-llm alignment for text-attributed graphs," 2025.

[52] X. Wang, Z. Zhang, L. Xiao, H. Chen, C. Ge, and W. Zhu, "Towards multimodal graph large language model," *Sci. China Inf. Sci.*, vol. 68, no. 11, pp. 1–20, 2025.

[53] X. Zhu, H. Xue, Z. Zhao, W. Xu, J. Huang, M. Guo, Q. Wang, K. Zhou, I. Razzak, and Y. Zhang, "Llm as gnn: Graph vocabulary learning for text-attributed graph foundation models," *arXiv preprint arXiv:2503.03313*, 2025.

[54] H. Zhou, J. Du, C. Zhou, C. Yang, Y. Xiao, Y. Xie, and X. Huang, "Each graph is a new language: Graph learning with llms," in *Findings of ACL*, 2025, pp. 17 548–17 559.

[55] Y. Fang, D. Fan, D. Zha, and Q. Tan, "Gaugllm: Improving graph contrastive learning for text-attributed graphs with large language models," in *KDD*, 2024, pp. 747–758.

[56] H. Chen, X. Wang, J. Chao, L. Feng, and W. Zhu, "A unified graph language model for multi-domain multi-task graph alignment instruction tuning," *arXiv preprint arXiv:2605.12197*, 2026.

[57] X. He, X. Bresson, T. Laurent, A. Perold, Y. LeCun, and B. Hooi, "Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning," in *ICLR*, 2023.

[58] G. Li, M. Müller, A. K. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?" in *International Conference on Computer Vision*, 2019, pp. 9266–9275.

[59] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Trans. Graph.*, vol. 38, no. 5, pp. 146:1–146:12, 2019.

[60] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual review of sociology*, vol. 27, no. 1, pp. 415–444, 2001.

[61] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1225–1234.

[62] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[63] Q. Li, Z. Zhu, and G. Tang, "Alternating minimizations converge to second-order optimal solutions," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3935–3943.

[64] Z. Deng, Y. Dong, S. Zhang, and J. Zhu, "Understanding and exploring the network with stochastic architectures," in *NeurIPS*, 2020.

[65] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.

[66] S. Lee, S. Purushwalkam, M. Cogswell, D. J. Crandall, and D. Batra, "Why M heads are better than one: Training a diverse ensemble of deep networks," *CoRR*, vol. abs/1511.06314, 2015.

[67] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *NeurIPS*, vol. 33, pp. 22 118–22 133, 2020.

[68] P. Mernyei and C. Cangea, "Wiki-cs: A wikipedia-based benchmark for graph neural networks," *arXiv preprint arXiv:2007.02901*, 2020.

[69] M. Newman, *Networks*. Oxford university press, 2018.

[70] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[71] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.

[72] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR*, 2020.

[73] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.

[74] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1761–1770.

[75] A. Noy, N. Nayman, T. Ridnik, N. Zamir, S. Doveh, I. Friedman, R. Giryes, and L. Zelnik, "Asap: Architecture search, anneal and prune," in *International Conference on Artificial Intelligence and Statistics*, 2020, pp. 493–503.

[76] N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik, "Xnas: Neural architecture search with expert advice," in *NeurIPS*, 2019, pp. 1975–1985.

[77] C. Guan, X. Wang, H. Chen, Z. Zhang, and W. Zhu, "Large-scale graph neural architecture search," in *International Conference on Machine Learning*. PMLR, 2022, pp. 7968–7981.

[78] J. Chen, J. Gao, Z. Wu, R. Al-Sabri, and B. M. Oloulade, "Decoupled differentiable graph neural architecture search," *Information Sciences*, vol. 673, p. 120700, 2024.

[79] D. Yu, R. Zhang, Z. Jiang, Y. Wu, and Y. Yang, "Graph-revised convolutional network," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part III*. Springer, 2021, pp. 378–393.

[80] Q. Wu, W. Zhao, Z. Li, D. Wipf, and J. Yan, "Nodeformer: A scalable graph structure learning transformer for node classification," in *NeurIPS*, 2022.

[81] L. Wu, H. Lin, Z. Liu, Z. Liu, Y. Huang, and S. Z. Li, "Homophily-enhanced self-supervision for graph structure learning: Insights and directions," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 9, pp. 12 358–12 372, 2024.

[82] Y. Wu, J. Yao, X. Xia, J. Yu, R. Wang, B. Han, and T. Liu, "Mitigating label noise on graph via topological sample selection," *arXiv preprint arXiv:2403.01942*, 2024.

[83] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.

[84] Y. Li, P. Wang, X. Zhu, A. Chen, H. Jiang, D. Cai, V. W. K. Chan, and J. Li, "Glbench: A comprehensive benchmark for graph with large language models," *arXiv preprint arXiv:2407.07457*, 2024.

[85] Z. Chen, H. Mao, H. Li, W. Jin, H. Wen, X. Wei, S. Wang, D. Yin, W. Fan, H. Liu *et al.*, "Exploring the potential of large language models (llms) in learning on graphs," *ACM SIGKDD Explorations Newsletter*, vol. 25, no. 2, pp. 42–61, 2024.

[86] H. Liu, J. Feng, L. Kong, N. Liang, D. Tao, Y. Chen, and M. Zhang, "One for all: Towards training one graph model for all classification tasks," in *ICLR*, 2024.
[87] X. Huang, K. Han, Y. Yang, D. Bao, Q. Tao, Z. Chai, and Q. Zhu, "Can gnn be good adapter for llms?" in *Proceedings of the ACM on Web Conference 2024*, 2024, pp. 893–904.
[88] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 3982–3992.
[89] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.
[90] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.
[91] E. Chien, W.-C. Chang, C.-J. Hsieh, H.-F. Yu, J. Zhang, O. Milenkovic, and I. S. Dhillon, "Node feature extraction by self-supervised multi-scale neighborhood prediction," in *ICLR*, 2022.
[92] Y. Zhu, Y. Wang, H. Shi, and S. Tang, "Efficient tuning and inference for large language models on textual graphs," *arXiv preprint arXiv:2401.15569*, 2024.
[93] J. Zhao, L. Zhuo, Y. Shen, M. Qu, K. Liu, M. Bronstein, Z. Zhu, and J. Tang, "Graphtext: Graph reasoning in text space," *arXiv preprint arXiv:2310.01089*, 2023.
[94] R. Chen, T. Zhao, A. Jaiswal, N. Shah, and Z. Wang, "Llaga: Large language and graph assistant," *arXiv preprint arXiv:2402.08170*, 2024.
[95] J. Zhao, M. Qu, C. Li, H. Yan, Q. Liu, R. Li, X. Xie, and J. Tang, "Learning on large-scale text-attributed graphs via variational inference," in *ICLR*, 2023.
[96] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.
[97] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence,*, 2019, pp. 6246–6250.
[98] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *ICLR*, 2019.
[99] N. Chen, Y. Li, J. Tang, and J. Li, "Graphwiz: An instruction-following language model for graph problems," *arXiv preprint arXiv:2402.16029*, 2024.

Xin Wang is currently an Associate Professor at the Department of Computer Science and Technology, Tsinghua University. He got both of his Ph.D. and B.E degrees in Computer Science and Technology from Zhejiang University, China. He also holds a Ph.D. degree in Computing Science from Simon Fraser University, Canada. His research interests include multimedia intelligence, machine learning and data mining. He has published over 250 high-quality research papers in ICML, NeurIPS, IEEE TPAMI, IEEE TKDE, ACM TOIS, ACM KDD, WWW, ACM SIGIR, ACM Multimedia etc., winning four best paper awards including IEEE ICME and ACM Multimedia Asia. He is the recipient of ACM China Rising Star Award, IEEE TCMC Rising Star Award and DAMO Academy Young Fellow.

Haibo Chen is currently a Ph.D. student at the Department of Computer Science and Technology, Tsinghua University. He received his B.E. degree from the School of Computer Science and Engineering, Central South University. His main research interests include graph machine learning, out-of-distribution learning, and multi-modal graphs.

Linxin Xiao is currently a master's student at the Department of Computer Science and Technology, Tsinghua Universiy, where he also received his B.E. degree. His research interests lie in graph machine learning, molecular relational learning, and multi-modal graphs.

Zeyang Zhang received his B.E. from the Department of Computer Science and Technology, Tsinghua University in 2020. He is a Ph.D. candidate in the Department of Computer Science and Technology of Tsinghua University. His main research interests focus on graph representation learning, automated machine learning and out-of-distribution generalization. He has published several papers in prestigious conferences, *e.g.*, NeurIPS, AAAI, etc.

Wenwu Zhu is currently a Professor in the Department of Computer Science and Technology at Tsinghua University. He also serves as the Vice Dean of Beijing National Research Center for Information Science and Technology.

His research interests include graph machine learning, curriculum learning, data-driven multimedia, big data. He has published over 400 referred papers, and is inventor of over 100 patents. He received ten Best Paper Awards, including ACM Multimedia 2012 and IEEE Transactions on Circuits and Systems for Video Technology in 2001 and 2019.

He serves as the EiC for IEEE Transactions on Circuits and Systems for Video Technology, the EiC for IEEE Transactions on Multimedia (2017-2019) and the Chair of the steering committee for IEEE Transactions on Multimedia (2020-2022). He serves as General Co-Chair for ACM Multimedia 2018 and ACM CIKM 2019. He is an AAAS Fellow, IEEE Fellow, ACM Fellow, SPIE Fellow, and a member of Academia Europaea.