# Large-Scale Graph Neural Architecture Search

**Chaoyu Guan** [1] **Xin Wang** [1] **Hong Chen** [1] **Ziwei Zhang** [1] **Wenwu Zhu** [1]

## Abstract

Graph Neural Architecture Search (GNAS) has become a powerful method in automatically discovering suitable Graph Neural Network (GNN) architectures for different tasks. However, existing approaches fail to handle large-scale graphs because current performance estimation strategies in GNAS are computationally expensive for large-scale graphs and suffer from consistency collapse issues. To tackle these problems, we propose the **G**raph **A**rchitect**U**re **S**earch at **S**cale (GAUSS) method that can handle large-scale graphs by designing an efficient light-weight supernet and the joint architecture-graph sampling. In particular, a graph sampling-based single-path one-shot supernet is proposed to reduce the computation burden. To address the consistency collapse issues, we further explicitly consider the joint architecture-graph sampling through a novel architecture peer learning mechanism on the sampled sub-graphs and an architecture importance sampling algorithm. Our proposed framework is able to smooth the highly non-convex optimization objective and stabilize the architecture sampling process. We provide theoretical analyses on GAUSS and empirically evaluate it on five datasets whose vertex sizes range from $10^4$ to $10^8$. The experimental results demonstrate substantial improvements of GAUSS over other GNAS baselines on all datasets. To the best of our knowledge, the proposed GAUSS method is the first graph neural architecture search framework that can handle graphs with billions of edges within 1 GPU day[1].

## 1. Introduction

Graph Neural Networks (GNNs) become more and more popular for their abilities to analyze structural relation data in various real-world applications such as recommender systems (Zhu et al., 2015; Yang et al., 2021; Cai et al., 2022), knowledge graphs (Vrandecic & Krötzsch, 2014; Lovelace et al., 2021; Cao et al., 2021), medical property prediction (Szklarczyk et al., 2019; Wishart et al., 2018), image processing (Yang et al., 2019), etc. To customize GNN architectures for diverse downstream tasks, more and more research attentions have been paid to develop Graph Neural Architecture Search (GNAS) methods (Gao et al., 2020; Li et al., 2021; Zhao et al., 2021; Guan et al., 2021a) in order to automatically search for the optimal GNN architectures. Architectures of many recent GNNs are thus automatically designed via GNAS to further benefit downstream tasks. However, current GNAS methods can only handle graphs at a relatively small scale with at most million-scale nodes and edges. In real-world scenarios, many applications require handling large-scale graphs with billion-scale nodes and edges (Lovelace et al., 2021; Hu et al., 2020). Existing GNAS methods fail to handle such large-scale graphs because of the severe scalability issues.

In this paper, we study Graph Neural Architecture Search (GNAS) for large-scale graphs. The bottleneck of the existing GNAS approaches in terms of scalability is that they depend on computationally expensive performance estimation strategies with a high time/space complexity for large-scale graphs. For example, Gao et al.'s (2020) work needs to train every single model until convergence to derive accurate rewards for the Reinforcement Learning (RL) controller, Li et al.'s (2021) and Zhao et al.'s (2021) works require calculating the output of every operation within the supernet on the whole graph to estimate reliable architecture parameters. All these performance estimation strategies in existing works fail to handle large-scale graphs, e.g., graphs with billion-scale nodes and edges. A straight-forward approach to solve the above issues is leveraging graph sampling techniques (Chiang et al., 2019; Hamilton et al., 2017; Zeng et al., 2020). However, directly applying these techniques to the current supernet training in GNAS will result in consistency collapse issues. Specifically, if we train the supernet by independently adopting an architecture sampling and a graph sampling process, the supernet will have poor rank-

[1]Media and Network Lab, Department of Computer Science and Technology, Tsinghua University. Correspondence to: Wenwu Zhu <wwzhu@tsinghua.edu.cn>, Xin Wang <xin_wang@tsinghua.edu.cn>.

*Proceedings of the $39^{th}$ International Conference on Machine Learning*, Baltimore, Maryland, USA, PMLR 162, 2022. 

[1]Code is available at https://www.github.com/THUMNLab/GAUSS.

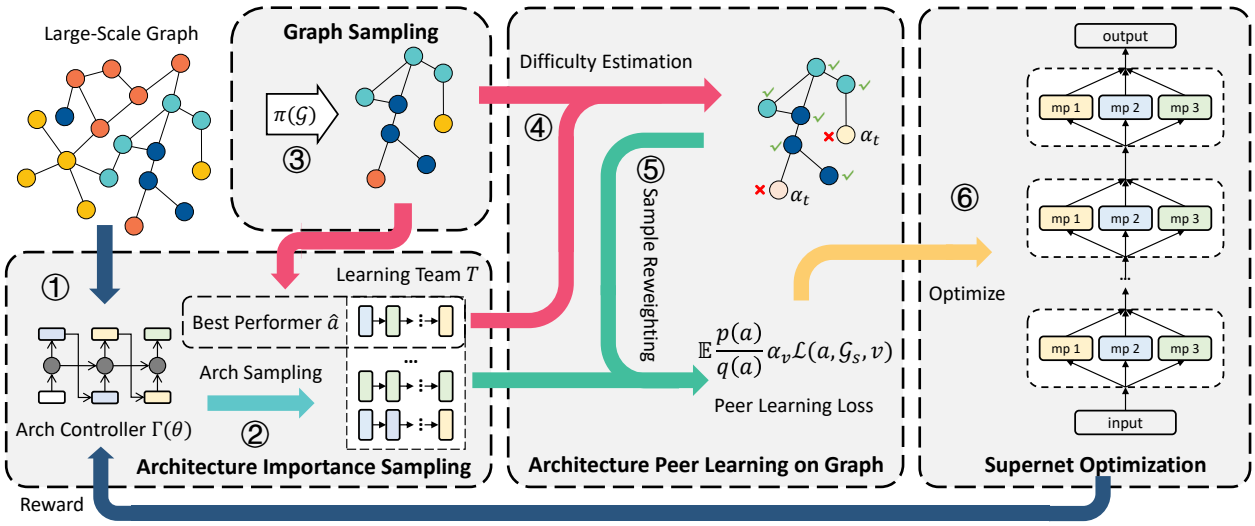


*Figure 1.* **The joint architecture-graph sampling based supernet training in GAUSS**. In each iteration, we ① first learn a proposal distribution using an architecture controller on the supernet, and then ② sample a learning team $T$ from the architecture controller $\gamma(\theta)$. We next ③ sample a sub-graph from the original large-scale graph following $\pi(\mathcal{G})$, and ④ leverage the best performer, i.e., architecture with the best performance, in team $T$ to estimate the node difficulties in the sampled subgraph in order to smooth the optimization objective. The smoothed objective ⑤ is then used to estimate the architecture peer learning loss of the whole team, followed by the supernet optimization ⑥. The performance of the optimized supernet on the large-scale graph will act as the reward to guide the training of the architecture controller in the next optimization cycle.

ing correlations between the estimated performances and ground truth performances. For example, we only observe a 0.2 Kendall's $\tau$ correlation, which is far from satisfaction (see Section 5 for details). To solve the computation challenge and avoid consistency collapse issues, in this paper, we propose **G**raph **A**rchitect**U**re **S**earch at **S**cale (GAUSS) method to automatically search GNN architectures for large-scale graphs. As shown in Figure 1, we address the computation challenge by designing an efficient light-weight supernet, and explicitly consider the joint sampling between architectures and graph to avoid the consistency collapse issues. In particular, we build a supernet based GNN performance estimator to efficiently estimate the performances of architectures through sharing all the model parameters in the search space, and utilize graph sampling methods and single-path formulation to train the supernet on large-scale graphs. Such a sampling based supernet training paradigm can enable the time/space cost to be the same as training a single GNN architecture over the original graph. To explicitly consider the joint sampling between architecture and graph, we propose a novel architecture peer learning mechanism that encourages the architectures with currently the best performance to smooth optimization objectives and gradually converges to the original optimization goal. To further stabilize the supernet optimization process, we explicitly optimize the architecture distributions based on current performances of the supernet through reinforcement learning and importance sampling. Moreover, we propose to jointly sample architectures and graphs by iteratively adjusting the optimization objectives and the architecture distributions. This joint architecture-graph sampling strategy is able to generate a supernet that has better ranking correlations across candidate architectures with respect to the ground truth model performances, thus tackling the consistency collapse issues.

We empirically evaluate GAUSS on five datasets whose node sizes range from $10^4$ to $10^8$. Experimental results demonstrate that our proposed GAUSS model significantly outperforms other GNAS baselines in terms of both accuracy and efficiency. Detailed ablation studies further validate the design of our proposed method, including the architecture peer learning on graphs and the architecture importance sampling. We summarize our main contributions as follows:

- We propose **G**raph **A**rchitect**U**re **S**earch at **S**cale (GAUSS) method that can automatically search neural architectures for large-scale graphs with billion-scale edges. To the best of our knowledge, GAUSS is the first graph neural architecture search method capable of processing graphs with more than one billion edges within 1 GPU day.
- We propose a joint architecture-graph sampling strategy between graphs and neural architectures to avoid consistency collapse issues by smoothing the optimization process and stabilizing the supernet training on large-scale graphs.
- We theoretically analyze the correlation property of the architecture importance sampling in GAUSS. Exten-

sive experiments on five datasets with different sizes show that our GAUSS model substantially outperforms GNAS baselines and further enhance the hand-crafted models, demonstrating its superiority against state-of-the-art baselines.

## 2. Related Works

### 2.1. Graph Neural Networks at Scale

The current de facto standard design of graph neural networks mostly follows the message passing framework (Kipf & Welling, 2017; Velickovic et al., 2018; Hamilton et al., 2017; Gilmer et al., 2017). To solve the scalability issues of training GNNs, many graph sampling methods are proposed to train GNNs only on parts of the graphs. To name a few, GraphSAGE (Hamilton et al., 2017), FastGCN (Chen et al., 2018a), and VR-GCN (Chen et al., 2018b) propose to sample neighbors during each message-passing layer. GraphSAINT (Zeng et al., 2020) and ClusterGCN (Chiang et al., 2019) propose to sample sub-graphs to increase the connection density within the sampled batch.

However, all the sampling methods above are merely designed to scale and stabilize the training of individual GNNs, which cannot handle the much more complex scenarios during training GNAS supernets. Instead, we propose a joint architecture-graph sampling framework that gradually adjusts the optimization objective and explicitly minimizes the variance for GNAS at scale.

There also exists some related works (Huang et al., 2021; Wang, 2021) that improve the performance of GNNs on large-scale graphs through pre-processing, post-processing, or other techniques during training, which is orthogonal to our work since our main focus is to find a better GNN architecture. We show in the ablation study (see Section 5.3) that these techniques can be leveraged to further boost the searched GNNs.

### 2.2. Graph Neural Architecture Search

To automate the design of GNNs, many efforts (Qin et al., 2021; Wang et al., 2022; Guan et al., 2021b) have been made to leverage recent advances of NAS. Gao et al. (2020) design the first NAS search space customized for GNNs. They leverage Reinforcement Learning (RL) (Zoph & Le, 2017) to search for the optimal architectures. Li et al. (2020) propose to gradually reduce the search space to ease the search process. Zhao et al. (2021) propose a differentiable method and a transfer paradigm to search on a small subgraph and transfer back to the original graph. We refer the reader to the dedicated survey on GNAS (Zhang et al., 2021) for more details.

All of the existing works conduct their search process on relatively small-scale graphs, depending on the full-batch GNN training schema. The most relevant work with ours is EGAN (Zhao et al., 2021), which scales to moderate-scale graphs by sampling a small sub-graph as a proxy and directly searching on it. However, due to the complexity of the graph structures and the randomness of sampling, only searching on a small sub-graph cannot guarantee that the searched architectures suit the original graph. Besides, the minimal sampling size required by EGAN ($15\%$ nodes of the original graph) is still too computationally-expensive for large-scale graphs with billions of nodes and edges. Instead, our proposed method can efficiently and effectively search architectures for large-scale graphs.

**Algorithm 1** A Basic Version of Scalable Supernet Training

**Input:** The number of epochs $T_{total}$, the architecture space $\mathcal{A}$, the graph $\mathcal{G}$, the graph sampler $\pi(\mathcal{G})$, the learning rate $\gamma$.
$\texttt{SNet} \leftarrow \texttt{Initialize\_Supernet}(\mathcal{A}, \mathcal{G})$
**for** $t \in \{1, ..., T_{total}\}$ **do**
  Sample $\mathcal{G}_s$ according to $\pi(\mathcal{G})$
  Sample $a$ randomly from $\mathcal{A}$
  $\mathcal{L} = \texttt{CE}(\texttt{SNet}(a, \mathcal{G}_s, \mathcal{V}_s), \text{Label}_s)$
  $\mathbf{w} \leftarrow \mathbf{w} - \gamma \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$
**end for**
**Ouput:** SNet

## 3. Architecture Search for Large-Scale Graphs

To efficiently scale current graph neural architecture search frameworks to large-scale graphs, we resort to the one-shot formulation (Bender et al., 2018) to construct a supernet, and transform the scalability challenge into training the supernet on large-scale graphs. In this section, we introduce a basic formulation of scalable GNAS, while our full proposed method will be introduced in Section 4. We describe the one-shot graphnas in Section 3.1, and the sampling based GNAS supernet training in Section 3.2.

### 3.1. One-shot GNAS

The original goal of GNAS can be written as a bi-level optimization problem (Liu et al., 2019):

$$\begin{aligned} a^* &= \text{argmax}_{a \in \mathcal{A}} \texttt{Acc}_{\text{valid}}(a, \mathcal{G}, \mathbf{w}^*(a)) \\ \mathbf{w}^*(a) &= \text{argmin}_{\mathbf{w}(a)} \mathcal{L}_{\text{train}}(a, \mathcal{G}, \mathbf{w}(a)), \end{aligned} \tag{1}$$

where $a$ is an architecture and $\mathcal{A}$ is the search space, $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ is the graph that contains a number of nodes $v \in \mathcal{V}$ and edges $e \in \mathcal{E}$ representing their relationships. $\mathbf{w}(a)$ is the learnable parameters determined by architecture $a$. $\mathbf{w}^*(a)$ stands for the best parameters for $a$. $a^*$ is the best architectures in the defined search space for $\mathcal{G}$. $\mathcal{L}_{\text{train}}(a, \mathcal{G}, \mathbf{w}(a))$ is the loss of architecture $a$ on the avail-

able graph data. For example, for the node classification task, $\mathcal{L}_{\text{train}}(a, \mathcal{G}, \mathbf{w}(a))$ depends on all the labeled nodes as follows:

$$\mathcal{L}_{\text{train}}(a, \mathcal{G}, \mathbf{w}(a)) = \mathbb{E}_{v \in \mathcal{V}_{\text{train}}} \mathcal{L}(a, \mathcal{G}, \mathbf{w}(a), v) \quad (2)$$

$$\mathcal{L}(a, \mathcal{G}, \mathbf{w}(a), v) = \texttt{CE}(\hat{y}_v, y_v), \quad (3)$$

where CE means the cross entropy between predicted distribution $\hat{y}_v$ and ground truth label $y_v$.

Directly solving Eq. (1) requires time-consuming optimizations over $\mathbf{w}(a)$ for each $a$, which is unacceptable. We thus leverage the one-shot paradigm (Bender et al., 2018) to transform the bi-level optimization problem into a two-step optimization through weight sharing (Pham et al., 2018):

$$a^* = \text{argmax}_{a \in \mathcal{A}} \texttt{Acc}_{\text{valid}}(a, \mathcal{G}, \mathbf{w}^*) \quad (4)$$

$$\mathbf{w}^* = \text{argmin}_{\mathbf{w}} \mathbb{E}_{a \in \mathcal{A}} \mathcal{L}_{\text{train}}(a, \mathcal{G}, \mathbf{w}), \quad (5)$$

where $\mathbf{w}$ denotes the learnable parameters of the supernet. For each operation in each GNN layer, we maintain only one set of parameters to reduce the optimization complexity.

After deriving the trained supernet, we leverage regularized evolution (Real et al., 2019) to solve Eq. (4).

### 3.2. Sampling-based Supernet Training

Although the one-shot paradigm can greatly reduce the optimization complexity, Eq. (5) still needs to estimate the training loss $\mathcal{L}_{\text{train}}$ over the full graph $\mathcal{G}$, which is infeasible due to the computational complexity. We thus leverage sampling techniques (Hamilton et al., 2017; Zeng et al., 2020; Chiang et al., 2019) to train the proposed supernet at scale[2].

$$\begin{aligned} \mathcal{L}_{\text{train}}(a, \mathcal{G}) &\approx \mathbb{E}_{\mathcal{G}_s \sim \pi(\mathcal{G})} \mathcal{L}_{\text{train}}(a, \mathcal{G}_s) \\ \mathcal{L}_{\text{train}}(a, \mathcal{G}_s) &= \mathbb{E}_{v \in \mathcal{V}_s} \mathcal{L}(a, \mathcal{G}_s, v), \end{aligned} \quad (6)$$

where $\pi(\mathcal{G})$ is a graph sampling distribution in previous works, and $\mathcal{V}_s$ stands for the labeled nodes in $\mathcal{G}_s$. The overall supernet optimization objective is then defined as:

$$\mathcal{L} \triangleq \mathbb{E}_{a \in \mathcal{A}, \mathcal{G}_s \sim \pi(\mathcal{G})} \mathbb{E}_{v \in \mathcal{V}_s} \mathcal{L}(a, \mathcal{G}_s, v). \quad (7)$$

We use Monte Carlo sampling to optimize the supernet weights $\mathbf{w}$. At each iteration of the training stage, we sample a part of the graph following $\pi(\mathcal{G})$ to limit the message flow, and an architecture is uniformly sampled from the search space to estimate the loss. Then, we optimize it using gradient descents. Note that, because we only optimize a single model during each iteration, all the graph sampling methods mentioned above for optimizing individual GNNs can be applied here. See Algorithm 1 for the pseudo code.

**Algorithm 2** GAUSS Supernet Training

**Input:** The number of epochs $T_{total}$, the architecture space $\mathcal{A}$, the graph $\mathcal{G}$, the graph sampler $\pi(\mathcal{G})$, the team size $n$, the learning rate $\gamma$.
`Controller` $\leftarrow$ `Initialize_Controller()`
`SNet` $\leftarrow$ `Initialize_Supernet`$(\mathcal{A}, \mathcal{G})$
**for** $t \in \{1, ..., T_{total}\}$ **do**
  $\alpha_t \leftarrow \alpha_{min} \times (1 - \frac{t}{T_{total}}) + \frac{t}{T_{total}}$
  **if** a controller update epoch **then**
    ▷ *Architecture Importance Sampling*
    Optimize the `Controller` according to Eq. (15)
  **end if**
  Sample $\mathcal{G}_s$ according to $\pi(\mathcal{G})$
  ▷ *Graph Peer Learning*
  `Acc` $\leftarrow -$`inf`
  `SNet.clear_gradient()`
  **for** $a$, $q(a)$ in `Controller.sample`$(n)$ **do**
    $\mathbf{out} =$ `SNet`$(a, \mathcal{G}_s, \mathcal{V}_s)$
    $\texttt{Acc}_a =$ `Cal_Acc`$(\mathbf{out}, \text{label}_s)$
    **if** $\texttt{Acc}_a >$ `Acc` **then**
      `Acc` $\leftarrow \texttt{Acc}_a$
      $\boldsymbol{\alpha}_s \leftarrow$ `Gen_Mask`$(\alpha_t, \mathbf{out}, \text{label}_s)$ # Eq. (10)
    **end if**
    $\hat{\mathcal{L}} = \frac{p(a)}{q(a)} \texttt{CE}(\mathbf{out}, \text{label}_s) \odot \boldsymbol{\alpha}_s$
    $\hat{\mathcal{L}}$`.backward()`
  **end for**
  $\mathbf{w} \leftarrow \mathbf{w} - \gamma$`Gradient`
**end for**
**Ouput:** `SNet`

## 4. Joint Architecture-Graph Sampling

Although the above straight-forward method can scale the supernet training to large-scale graphs, the consistency of the supernet tends to collapse because of a much more non-convex optimization objective and unstable training procedure induced by two independent sampling processes. To smooth the optimization objective and stabilize the training process, we propose a novel peer learning algorithm to first form a learning group, and then let the best learner decide a smoother learning objective for the group. We further propose to use importance sampling to reduce the variance during architecture sampling to form better learning groups.

### 4.1. Architecture Peer Learning on Graph

The previous graph sampling methods (Hamilton et al., 2017; Zeng et al., 2020) are designed for optimizing individual GNN architectures. When we consider a possibly huge set of candidate architectures, the objective becomes highly non-convex to optimize. To help smooth the learning

[2]We omit $\mathbf{w}$ from now on when there is no ambiguity.

process, we propose to let the architectures help each other to find proper optimization objectives, imitating the peer learning process in education (O'Donnell & King, 1999).

To be specific, for each sampled sub-graph, we first randomly sample $n$ architectures as a learning team $T$. We then determine a smoother optimization objective for the rest of the architectures by estimating the difficulty of each node via the best performer $\hat{a} \in T$. Specifically, we set a lower weight $\alpha_t$ to the losses of nodes which are misclassified severely by $\hat{a}$ so that the learning team can focus more on easier parts of the objective and gradually progress to the difficult parts. This is similar to the Proctor model in peer learning (Boud, 2001), where senior students ($\hat{a}$ in our case) pass their knowledge to junior students. We formulate the optimization objective at the $t$-th step of optimization as follows:

$$\hat{\mathcal{L}} = \mathbb{E}_{T\in\mathcal{A}^n,\mathcal{G}_s\sim\pi(\mathcal{G})}\mathbb{E}_{a\in T,v\in\mathcal{V}_s}\alpha_v\mathcal{L}(a,\mathcal{G}_s,v) \tag{8}$$

$$\hat{a} = \text{argmax}_{a\in T}\texttt{Acc}_{\text{train}}(a,\mathcal{G}_s) \tag{9}$$

$$\alpha_v = \begin{cases} \alpha_t & l(\hat{a},v)\neq y_v \text{ and } p(\hat{a},v)>\lambda \\ 1 & \text{Otherwise} \end{cases}, \tag{10}$$

where $l(\hat{a},v)$ and $p(\hat{a},v)$ denotes the predicted label and the corresponding probabilities of the architecture $\hat{a}$ on node $v$, $\mathcal{A}^n$ denotes the Cartesian product of $\mathcal{A}$, $\lambda$ is a hyper-parameter controlling the threshold, and $\alpha_t \leq 1$ is the adjusted weight. To gradually recover the original optimization objective, we set $\alpha_t$ as follows:

$$\alpha_t = \alpha_{min} \times (1 - \frac{t}{T_{total}}) + \frac{t}{T_{total}}, \tag{11}$$

where $\alpha_{min} < 1$ and $T_{total}$ is the total number of optimization steps. Therefore, $\alpha_t$ will tend to uniformly converge to 1 at the end of optimization.

Note that in the above method, we need to store all the computational graphs of $n$ sampled architectures, which is memory-expensive for large-scale graphs. To reduce the complexity, we further propose a strategy to dynamically record the difficulty estimation of the current best architectures. In this way, we only need to handle 1 computational graph at a time. See Algorithm 2 for the pseudo code.

### 4.2. Architecture Importance Sampling

Even with a smooth optimization objective, the training process still suffers instability issues induced by sampling a learning team from $\mathcal{A}^n$. To further stabilize the training of the supernet on large-scale graphs, we propose to use importance sampling to reduce the optimization variance.

When we optimize the weights $\mathbf{w}$ of the supernet following Eq. (5), to avoid overfitting, we should focus on the average accuracy over the validation dataset, which can be rewritten as follows[3]:

$$\begin{aligned}\texttt{Acc}(\mathcal{A}) &\triangleq \mathbb{E}_{a\in\mathcal{A}}\texttt{Acc}_{\text{valid}}(a) \\ &= \mathbb{E}_{a\sim\Gamma(\mathcal{A})}\frac{p(a)}{q(a)}\texttt{Acc}_{\text{valid}}(a),\end{aligned} \tag{12}$$

where $\Gamma(\mathcal{A})$ is a proposal distribution of architectures, $p(a)$ and $q(a)$ are the probabilities given by the uniform distribution and $\Gamma(\mathcal{A})$. Using the importance sampling theory, we have the following property for the optimal proposal distribution:

**Theorem 4.1.** *The estimator of* $\texttt{Acc}(\mathcal{A})$ *in Eq. (12) reaches its minimal variance iff.* $q(a) \propto \texttt{Acc}_{\text{valid}}(a)$

*Proof.* See Appendix A for the detailed proof. □

However, directly calculating the optimal proposal distribution is intractable because we need to traverse all possible architectures $a$ in the search space $\mathcal{A}$. We thus propose an alternative approximation method based on reinforcement learning. Following the NAS literature (Zoph & Le, 2017; Gao et al., 2020), we formulate architecture generation as a Markov Decision Process (MDP). We build a controller based on GRU (Cho et al., 2014) to simulate the sequential layer operation choices:

$$\begin{aligned}\mathbf{h}_0 &= \mathbf{0},\ x_0 = \texttt{<bos>} & \\ \mathbf{h}_l &= \texttt{GRU}(\texttt{Emb}(x_{l-1}),\mathbf{h}_{l-1}) & l\in\{1,...,L\} \\ q(x_l|x_{0:l-1}) &= \texttt{Softmax}(\mathbf{W}\mathbf{h}_l) & l\in\{1,...,L\} \\ x_l &= \texttt{Sample}(q(x_l|x_{0:l-1})) & l\in\{1,...,L\}\end{aligned}$$

where $\mathbf{h}_0 \in \mathbb{R}^d$ are initial hidden states, $x_0$ is the dummy layer choice standing for the beginning of the sequence, $\mathbf{h}_l \in \mathbb{R}^d$ and $x_l$ with $l \in \{1,...,L\}$ represent the hidden state and choice of operations for the time stamp and layer $l$, $\mathbf{W} \in \mathbb{R}^{c\times d}$ is a learnable weight matrix that maps the hidden states $\mathbf{h}_l$ to the operation distribution in the $l$-th layer with $c$ operation choices. `GRU`, `Emb`, `Softmax`, and `Sample` are functions and parameterized modules.

The controller can be seen as a parameterized distribution $\Gamma(\theta)$. According to proposition 4.1, we choose the accuracy of the sampled architectures as the reward. The optimization objective of the GRU controller is defined as:

$$\theta = \text{argmax}_\theta\mathcal{R}(\theta) \triangleq \text{argmax}_\theta\mathbb{E}_{a\in\Gamma(\theta)}\texttt{Acc}_{\text{valid}}(a). \tag{13}$$

Following REINFORCE (Williams, 1992), we use policy

[3] We omit $\mathcal{G}$ for conciseness.

gradient to optimize the objective as follows:

$$\begin{aligned}\nabla R(\theta) &= \nabla \mathbb{E}_{a\sim\Gamma(\theta)} \texttt{Acc}_{\text{valid}}(a) \\ &= \sum_{a\in\mathcal{A}} \texttt{Acc}_{\text{valid}}(a) \nabla q(a,\theta) \\ &= \sum_{a\in\mathcal{A}} q(a,\theta) \texttt{Acc}_{\text{valid}}(a) \nabla \log(q(a,\theta)) \\ &= \mathbb{E}_{a\sim\Gamma(\theta)} \texttt{Acc}_{\text{valid}}(a) \nabla \log(q(a,\theta)).\end{aligned} \tag{14}$$

Note that the global optimal solution to Eq. (13) is a Dirac distribution that only has non-zero probability on the architecture(s) with the best accuracy, which is too discrete to be considered as a proper proposal distribution. To solve this problem, we add a regularizer term over the learned distribution. The optimization then becomes:

$$\theta = \text{argmax}_\theta(\mathcal{R}(\theta) + \beta\mathcal{H}(\Gamma(\theta))), \tag{15}$$

where $\mathcal{H}(\Gamma(\theta))$ stands for the entropy of the architecture distribution, and $\beta > 0$ is a hyper-parameter controlling the smoothness of the learned distribution. The global optimal solution to Eq. (15) satisfies the following property:

**Theorem 4.2.** *If the optimal solution $q^*(a)$ to Eq. (15) is injective, i.e., for architecture $a_i \neq a_j$, we have $q^*(a_i) \neq q^*(a_j)$, then the Kendall's $\tau$ correlation of the architecture probability defined by $q^*(a)$ and the architecture accuracy from supernet is 1, i.e., for any $a_i, a_j$, we have $(q^*(a_i) - q^*(a_j))(\texttt{Acc}_{\text{valid}}(a_i) - \texttt{Acc}_{\text{valid}}(a_j)) \geq 0$.*

*Proof.* Refer to Appendix B for the detailed proof. □

We then utilize the proposal distribution $\Gamma(\mathcal{A})$ to replace the uniform sampling process in Eq. (8). The objective function can be written as:

$$\hat{\mathcal{L}} = \mathbb{E}_{T\sim\Gamma^n(\mathcal{A}),\mathcal{G}_s\sim\pi(\mathcal{G})} \mathbb{E}_{a\in T, v\in\mathcal{V}_s} \frac{p(a)}{q(a)} \alpha_v \mathcal{L}(a, \mathcal{G}_s, v). \tag{16}$$

At each optimization step, we first derive a better proposal distribution $\Gamma(\mathcal{A})$, and then formulate a learning team and reweight the optimization objective. The pseudo code of the proposed method is shown in Algorithm 2.

## 5. Experiments

In this section, we empirically evaluate the proposed GAUSS on graph benchmark datasets with different scales to demonstrate its efficiency and effectiveness. We also conduct detailed ablation studies of the proposed architecture peer learning on graphs and the architecture importance sampling.

*Table 1.* Statistics of datasets

| DATASET | #NODES | #EDGES |
|---|---|---|
| CS | 18,333 | 81,894 |
| PHYSICS | 34,493 | 247,962 |
| ARXIV | 169,343 | 1,166,243 |
| PRODUCTS | 2,449,029 | 61,859,140 |
| PAPERS100M | 111,059,956 | 1,615,685,872 |

### 5.1. Datasets and Baselines

We select 5 node classification datasets from OGB (Hu et al., 2020) and GNN Benchmark (Shchur et al., 2018), including graphs of different scales to demonstrate the scalability of our proposed model. The statistics of the used datasets is listed in Table 1. Note that our largest dataset, Papers100M, contains over 100 million nodes and 1 billion edges. Refer to Appendix C for more details.

We select representative hand-crafted state-of-the-art GNN architectures for comparison, including Graph Convolutional Network (GCN) (Kipf & Welling, 2017), Graph Attention Network (GAT) (Velickovic et al., 2018), GraphSAGE (Hamilton et al., 2017), and Graph Isomorphism Network (GIN) (Xu et al., 2019). We also compare representative GNAS baselines, including GraphNAS (Gao et al., 2020), SGAS (Li et al., 2020), DARTS (Liu et al., 2019), and EGAN (Zhao et al., 2021).

Since the main focus of this paper is to search for the best GNN architectures, we only compare the performance of baselines based on their architectures and exclude all the sophisticated pre-processing or post-processing techniques and other tricks like Correct and Smooth (C&S) (Huang et al., 2021), Node2Vec (Grover & Leskovec, 2016), label reuse (Wang, 2021), self-knowledge distillation (Hinton et al., 2015), etc. In the ablation study, we show that our searched architectures can also benefit from these techniques.

### 5.2. Implementation Details

#### 5.2.1. SUPERNET CONSTRUCTION

We choose representative message-passing GNN layers to form our *operation pool*, namely GCNConv (Kipf & Welling, 2017), GATConv (Velickovic et al., 2018), SAGEConv (Hamilton et al., 2017), GINConv (Xu et al., 2019), GraphConv (Morris et al., 2019), and Linear. We build the supernet as a sequence of GNN layers, with each layer following a batch normalization, a ReLU activation, and a dropout layer. To ensure a fair comparison, we follow the best practice and use the same set of hyper-parameters for both the baselines and GAUSS. More details for repro-

*Table 2.* The results of our proposed method and baseline methods. We report both the validation and test accuracy [%] over 10 runs with different seeds. OOT means out-of-time (cannot converge within 1 single GPU day), while OOM means out-of-memory (cannot run on a Tesla V100 GPU with 32GB memory). The results of the best hand-crafted and automated method are in bold, respectively.

| Methods | CS | | Physics | | Arxiv | | Products | | Papers100M | |
|---|---|---|---|---|---|---|---|---|---|---|
| | valid | test | valid | test | valid | test | valid | test | valid | test |
| GCN | $94.10_{0.21}$ | $93.98_{0.21}$ | $96.29_{0.05}$ | $96.38_{0.07}$ | $72.76_{0.15}$ | $71.70_{0.18}$ | $91.75_{0.04}$ | $80.19_{0.46}$ | $70.32_{0.11}$ | $67.06_{0.17}$ |
| GAT | $93.74_{0.27}$ | $93.48_{0.36}$ | $96.25_{0.23}$ | $96.37_{0.23}$ | $73.19_{0.12}$ | $71.85_{0.21}$ | $90.75_{0.16}$ | $80.59_{0.40}$ | $70.26_{0.16}$ | $67.26_{0.06}$ |
| SAGE | $95.65_{0.07}$ | $95.33_{0.11}$ | $96.76_{0.10}$ | $96.72_{0.07}$ | $73.11_{0.08}$ | $71.78_{0.15}$ | $91.75_{0.04}$ | $80.19_{0.46}$ | $70.32_{0.11}$ | $67.06_{0.17}$ |
| GIN | $92.00_{0.43}$ | $92.14_{0.34}$ | $96.03_{0.11}$ | $96.04_{0.15}$ | $71.16_{0.10}$ | $70.01_{0.33}$ | $91.58_{0.10}$ | $79.07_{0.52}$ | $68.98_{0.16}$ | $65.78_{0.09}$ |
| GraphNAS | $94.90_{0.14}$ | $94.67_{0.23}$ | $96.76_{0.10}$ | $96.72_{0.07}$ | $72.76_{0.15}$ | $71.70_{0.18}$ | OOT | OOT | OOT | OOT |
| SGAS | $95.62_{0.06}$ | $95.44_{0.06}$ | $96.44_{0.10}$ | $96.50_{0.11}$ | $72.38_{0.11}$ | $71.34_{0.25}$ | OOM | OOM | OOM | OOM |
| DARTS | $95.62_{0.06}$ | $95.44_{0.06}$ | $96.21_{0.16}$ | $96.40_{0.21}$ | $73.43_{0.07}$ | $72.10_{0.25}$ | OOM | OOM | OOM | OOM |
| EGAN | $95.60_{0.10}$ | $95.43_{0.05}$ | $96.39_{0.18}$ | $96.45_{0.19}$ | $72.91_{0.25}$ | $71.75_{0.35}$ | OOM | OOM | OOM | OOM |
| Basic | $95.13_{0.07}$ | $95.45_{0.05}$ | $96.25_{0.06}$ | $96.53_{0.09}$ | $73.28_{0.08}$ | $72.06_{0.33}$ | $\mathbf{91.79}_{0.11}$ | $80.56_{0.39}$ | $69.49_{0.37}$ | $66.24_{0.46}$ |
| GAUSS | $\mathbf{96.08}_{0.11}$ | $\mathbf{96.49}_{0.11}$ | $\mathbf{96.79}_{0.06}$ | $\mathbf{96.76}_{0.08}$ | $\mathbf{73.63}_{0.10}$ | $\mathbf{72.35}_{0.21}$ | $91.60_{0.12}$ | $\mathbf{81.26}_{0.36}$ | $\mathbf{70.57}_{0.07}$ | $\mathbf{67.32}_{0.18}$ |

ducibility are provided in Appendix D and Appendix E.

#### 5.2.2. Training Details

For the architecture peer learning on graphs, we set the size of the team learning group $n$ to 10, and set $\alpha_{min}$ to 0.6 at the beginning of the search. At each controller optimization cycle, we train the GRU-based controller for 5 epochs and sample 12 architectures every epoch, using a moving average baseline with an update ratio 0.1. All the experiments are implemented using PyTorch and are conducted on a Tesla V100 GPU with 32GB of memory. The supernet training and search process of all methods are limited to 1 GPU day. Refer to Appendix D for more details.

### 5.3. Experiment Results

#### 5.3.1. Main Results

Table 2 shows the overall comparisons of GAUSS and previous hand-crafted and automated baselines on all 5 datasets. We also show the results of the basic version of scalable supernet training in Section 3. We can observe that the searched architectures of GAUSS consistently outperform the hand-crafted baselines, searched architectures, as well as the basic baseline, demonstrating the effectiveness of the proposed GAUSS method. Note that, for large-scale graphs with the node number exceeding $10^6$, i.e., Products and Papers100M, the current GNAS algorithms are not applicable for either time or memory reasons. Although the basic scalable baseline can run on these datasets, the searched results are not so competitive with hand-crafted baselines. In comparison, the proposed GAUSS can still give superior results when scaling to even larger datasets like papers100M with more than $10^8$ nodes and $10^9$ edges, showing the scalability, generality, and expressiveness of the proposed method.

*Table 3.* The results of ablation study on the functional components of the proposed GAUSS. We report Kendall's $\tau$ [%] correlation of different training strategies over the sampled architectures on CS and Physics. APLG stands for Architecture Peer Learning on Graphs. AIS stands for Architecture Importance Sampling.

| Strategy | CS | Physics |
|---|---|---|
| Basic (Uniform Sampling) | 23.71 | 20.35 |
| + APLG | 64.86 | 53.51 |
| + APLG + AIS (GAUSS) | **72.83** | **60.72** |

#### 5.3.2. Ablation Studies

To understand and analyze the functional parts of GAUSS, we further conduct detailed ablation studies on the influence of the proposed architecture peer learning on graphs and architecture importance sampling. We take CS and Arxiv as example ablation study datasets to demonstrate how these two parts help improve the consistency of the supernet training.

To be specific, we randomly sample 20 architectures from the search space, and record their validation accuracy on the corresponding datasets. Then, we perform different supernet training methods and compute the rank correlation (Kendall's $\tau$ correlation in this paper) on the sampled architectures.

Table 3 shows the $\tau$ value under different experimental settings. We can observe that the Architecture Peer Learning on Graph (APLG) contributes most to the consistency of the supernet, demonstrating the effectiveness of the proposed

*Table 4.* The ablation study on the interaction within the learning team. We report Kendall $\tau$ [%] under different $\alpha_{min}$ settings.

| $\alpha_{min}$ | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
|---|---|---|---|---|---|---|
| CS | 63.25 | 62.30 | 64.42 | **72.83** | 64.43 | 63.27 |
| Physics | 51.13 | 59.39 | 58.86 | **60.72** | 57.58 | 57.28 |

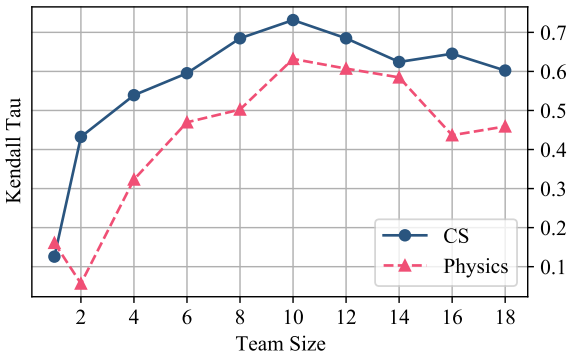


*Figure 2.* The ablation study on the influence of the learning team size. We report Kendall's $\tau$ correlation under different team sizes on CS and Physics.

team learning and interaction methods. Based on APLG, the Architecture Importance Sampling (AIS) can further improve the consistency, showing that a proper proposal distribution can also be beneficial.

We conduct further analysis on how the APLG and AIS help boosts the consistency of supernet. Specifically, we exam key hyper-parameters including the team size $n$ and the minimum weight $\alpha_{min}$ in APLG, and the accuracy estimation during supernet training through AIS.

**Influence of the Team Size**. Figure 2 shows the effect of the team size. We can observe that the Kendall's $\tau$ correlation gradually increases when there are more students in one learning team. A plausible reason is that, as the team size becomes larger, the optimization objective can be more properly adjusted. However, when the team size is larger than 10, the consistency of the supernet tends to stabilize or even begins to drop, which shows that the optimization objective tends to converge or be misled by the overfitted top performers on the sampled sub-graphs.

**Influence of Interaction within the Learning Team**. Table 4 shows the different $\alpha_{min}$ choices and their corresponding Kendall $\tau$ on the two datasets. We can observe that, the correlation will first increase as $\alpha_{min}$ increases, and then gradually drop after it reaches 0.6. Note that when $\alpha_{min} = 1$, the students in one learning group will have no interaction with each other. The results show that, in general, the interaction among students is beneficial to the supernet consistency because of a smoother optimization objective. However, punishing the hard nodes too much (a smaller $\alpha_{min}$ setting) will potentially lead to degenerated consistency because of too much information loss and the limitation of the best performers on identifying hard nodes.

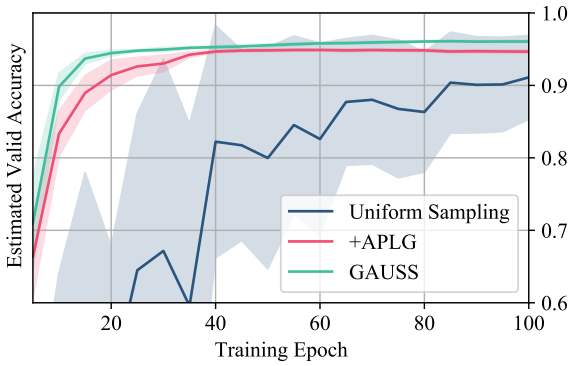


*Figure 3.* The ablation study on the variance reduction of architecture importance sampling. We report the estimated validation accuracy of sampled architectures at each training epoch of different training strategies, with their variance shown in the light color.

*Table 5.* The compatibility analysis of other techniques. We compare different techniques combination on both the hand-crafted GCN model and our GAUSS model. We report the accuracy [%] on the validation dataset of Arxiv.

| Models | GCN | GAUSS |
|---|---|---|
| Plain | 72.76 | **73.63** |
| + Node2Vec | 73.51 | **74.05** |
| + Node2Vec + label | 73.56 | **74.37** |
| + Node2Vec + label + C&S | 73.78 | **74.49** |
| + Node2Vec + label + C&S + KD | 73.96 | **74.55** |

**The Stability and Variance Reduction of AIS**. To better understand how AIS influences the supernet training process, we visualize the estimated expectation of accuracies over the whole architecture search space in Figure 3. We can see that the proposed AIS can indeed reduce the variance of the accuracy estimator, especially at the beginning of the training (e.g., in the first 40 epochs), thus leading to better consistency convergence of the supernet. In contrast, the supernet trained using uniform sampling and only using APLG still suffers from high variances, resulting in a relatively poor consistency.

**Compatibility with Other Techniques**. Although the focus of this paper is on the best GNN architecture, we show that our searched results can also benefit from some architecture-agnostic training techniques proposed specifically to boost the node classification performance. We report the performance of our models combined with four such techniques: Node2vec (Grover & Leskovec, 2016), Label Reuse (Wang,

2021), C&S (Huang et al., 2021), KD (Hinton et al., 2015). We report the results and compare it with GCN, where these techniques are originally used. The results on Arxiv are shown in Table 5. We can observe that the architecture discovered by GAUSS consistently outperforms hand-crafted GCN when combined with all these techniques.

## 6. Conclusion and Future Work

In this paper, we propose a scalable graph neural architecture search framework GAUSS to automatically search architectures for large-scale graphs. We propose a graph sampling based supernet training paradigm to allievate the computation burden, and develop a joint architecture-graph sampling methods to avoid consistency collapse problems. To the best of our knowledge, GAUSS is the first graph neural architecture search framework that can handle over $10^8$ nodes and $10^9$ edges in 1 GPU day. Future works include evaluating GAUSS on other large-scale graphs and other graph tasks like graph classification and link prediction.

## Acknowledgement

This work is supported by the National Key Research and Development Program of China No. 2020AAA0106300 and National Natural Science Foundation of China No. 62102222.

## References

Bender, G., Kindermans, P., Zoph, B., Vasudevan, V., and Le, Q. V. Understanding and simplifying one-shot architecture search. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 549–558. PMLR, 2018.

Boud, D. Making the move to peer learning. *Peer Learning in Higher Education: Learning from and with Each Other*, pp. 1–17, 01 2001.

Cai, J., Wang, X., Guan, C., Tang, Y., Xu, J., Zhong, B., and Zhu, W. Multimodal continual graph learning with neural architecture search. In *Proceedings of the ACM Web Conference 2022*, WWW '22, pp. 1292–1300, 2022.

Cao, Z., Xu, Q., Yang, Z., Cao, X., and Huang, Q. Dual quaternion knowledge graph embeddings. In *AAAI*, pp. 6894–6902, 2021.

Chen, J., Ma, T., and Xiao, C. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net, 2018a.

Chen, J., Zhu, J., and Song, L. Stochastic training of graph convolutional networks with variance reduction. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 941–949. PMLR, 2018b.

Chiang, W., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019*, pp. 257–266. ACM, 2019.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014*, pp. 1724–1734. ACL, 2014.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Gao, Y., Yang, H., Zhang, P., Zhou, C., and Hu, Y. Graph neural architecture search. In Bessiere, C. (ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pp. 1403–1409. ijcai.org, 2020.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, 2017.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 855–864. ACM, 2016.

Guan, C., Wang, X., and Zhu, W. Autoattend: Automated attention representation search. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, volume 139 of *Proceedings of Machine Learning Research*, pp. 3864–3874. PMLR, 2021a.

Guan, C., Zhang, Z., Li, H., Chang, H., Zhang, Z., Qin, Y., Jiang, J., Wang, X., and Zhu, W. AutoGL: A library for automated graph learning. In *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*, 2021b. URL https://openreview.net/forum?id=0yHwpLeInDn.

Hamilton, W. L., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances*

*in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pp. 1024–1034, 2017.

Hinton, G. E., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*, 2020.

Huang, Q., He, H., Singh, A., Lim, S., and Benson, A. R. Combining label propagation and simple models outperforms graph neural networks. In *9th International Conference on Learning Representations, ICLR 2021*. OpenReview.net, 2021.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, 2017.

Li, G., Qian, G., Delgadillo, I. C., Müller, M., Thabet, A. K., and Ghanem, B. SGAS: sequential greedy architecture search. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020*, pp. 1617–1627. Computer Vision Foundation / IEEE, 2020.

Li, Y., Wen, Z., Wang, Y., and Xu, C. One-shot graph neural architecture search with dynamic search space. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, pp. 8510–8517. AAAI Press, 2021.

Liu, H., Simonyan, K., and Yang, Y. DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net, 2019.

Lovelace, J., Newman-Griffis, D., Vashishth, S., Lehman, J. F., and Rosé, C. P. Robust knowledge graph completion with stacked convolutions and a student re-ranking network. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021,*, pp. 1016–1029. Association for Computational Linguistics, 2021.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, pp. 4602–4609. AAAI Press, 2019.

O'Donnell, A. M. and King, A. *Cognitive Perspectives on Peer Learning*. Routledge, 1999.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4092–4101. PMLR, 2018.

Qin, Y., Wang, X., Zhang, Z., and Zhu, W. Graph differentiable architecture search with structure learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021*, 2021.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, pp. 4780–4789, 2019.

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

Szklarczyk, D., Gable, A. L., Lyon, D., Junge, A., Wyder, S., Huerta-Cepas, J., Simonovic, M., Doncheva, N. T., Morris, J. H., Bork, P., Jensen, L. J., and von Mering, C. STRING v11: protein-protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic Acids Res.*, 47(Database-Issue):D607–D613, 2019.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net, 2018.

Vrandecic, D. and Krötzsch, M. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014.

Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., and Zhang, Z. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Wang, X., Zhang, Z., and Zhu, W. Automated graph machine learning: Approaches, libraries and directions. *CoRR*, abs/2201.01288, 2022.

Wang, Y. Bag of tricks for node classification with graph neural networks. *CoRR*, abs/2103.13355, 2021.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256, 1992.

Wishart, D. S., Feunang, Y. D., Guo, A. C., Lo, E. J., Marcu, A., Grant, J. R., Sajed, T., Johnson, D., Li, C., Sayeeda, Z., Assempour, N., Iynkkaran, I., Liu, Y., Maciejewski, A., Gale, N., Wilson, A., Chin, L., Cummings, R., Le, D., Pon, A., Knox, C., and Wilson, M. Drugbank 5.0: a major update to the drugbank database for 2018. *Nucleic Acids Res.*, 46(Database-Issue):D1074–D1082, 2018.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net, 2019.

Yang, Z., Xu, Q., Zhang, W., Cao, X., and Huang, Q. Split multiplicative multi-view subspace clustering. *IEEE Trans. Image Process.*, 28(10):5147–5160, 2019.

Yang, Z., Xu, Q., Cao, X., and Huang, Q. Task-feature collaborative learning with application to personalized attribute prediction. *IEEE Trans. Pattern Anal. Mach. Intell.*, 43(11):4094–4110, 2021.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. K. Graphsaint: Graph sampling based inductive learning method. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net, 2020.

Zhang, Z., Wang, X., and Zhu, W. Automated machine learning on graphs: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021*, pp. 4704–4712. ijcai.org, 2021.

Zhao, H., Wei, L., quanming yao, and He, Z. Efficient graph neural architecture search, 2021. URL https://openreview.net/forum?id=IjIzIOkK2D6.

Zhu, W., Cui, P., Wang, Z., and Hua, G. Multimedia big data computing. *IEEE MultiMedia*, 22(03):96–c3, jul 2015. ISSN 1941-0166. doi: 10.1109/MMUL.2015.66.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, 2017.

## A. Proof of Theorem 4.1

*Proof.* We aim to explicitly minimize the estimator $\frac{p(a)}{q(a)}\texttt{Acc}_{\text{valid}}(a)$ through applying proper $q(a)$:

$$\underset{\Gamma(\mathcal{A})}{\text{minimize}} \ \mathbb{V}_{\Gamma(\mathcal{A})}\frac{p(a)}{q(a)}\texttt{Acc}_{\text{valid}}(a) \tag{17}$$

$$\text{subject to} \ \sum_{a\in\mathcal{A}} q(a) = 1. \tag{18}$$

For the minimization objective, we have:

$$\mathbb{V}_{a\in\Gamma(\mathcal{A})}\frac{p(a)}{q(a)}\texttt{Acc}_{\text{valid}}(a) = \sum_{a\in\mathcal{A}} q(a)\left(\frac{p(a)}{q(a)}\texttt{Acc}_{\text{valid}}(a) - \frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}}\texttt{Acc}_{\text{valid}}(a)\right)^2 \tag{19}$$

$$= \sum_{a\in\mathcal{A}} q(a)\left(\texttt{Acc}_{\text{valid}}(a)\frac{p(a)}{q(a)}\right)^2 - \left(\frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}}\texttt{Acc}_{\text{valid}}(a)\right)^2 \tag{20}$$

$$= \frac{1}{|\mathcal{A}|^2}\sum_{a\in\mathcal{A}}\frac{\texttt{Acc}^2_{\text{valid}}(a)}{q(a)} - \left(\frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}}\texttt{Acc}_{\text{valid}}(a)\right)^2. \tag{21}$$

We introduce the Lagrange Multipliers $\zeta(\zeta \neq 0)$ to handle the condition.

$$L(\Gamma(\mathcal{A}),\zeta) = \frac{1}{|\mathcal{A}|^2}\sum_{a\in\mathcal{A}}\frac{\texttt{Acc}^2_{\text{valid}}(a)}{q(a)} - \left(\frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}}\texttt{Acc}_{\text{valid}}(a)\right)^2 + \zeta\left(\sum_{a\in\mathcal{A}} q(a) - 1\right) \tag{22}$$

$$\frac{\partial L}{\partial \zeta} = \sum_{a\in\mathcal{A}} q(a) - 1 = 0 \tag{23}$$

$$\frac{\partial L}{\partial q(a)} = -\frac{\texttt{Acc}^2_{\text{valid}}(a)}{|\mathcal{A}|^2 q^2(a)} + \zeta = 0, \ \text{for } a \in \mathcal{A}. \tag{24}$$

Eq. (24) tell us that the best solution with minimal variance satisfy $q(a) \propto \texttt{Acc}_{\text{valid}}(a)$. □

## B. Proof of Theorem 4.2

*Proof.* We can prove by contradiction. Suppose we have reached the optimal distribution $q^*(a)$ which maximizes $\mathcal{R}(\theta) + \beta\mathcal{H}(\Gamma(\theta))$, and there are two architectures $a_i$, $a_j$ that do not meet the conclusion of Theorem 4.2, i.e.:

$$(q^*(a_i) - q^*(a_j))(\texttt{Acc}_{\text{valid}}(a_i) - \texttt{Acc}_{\text{valid}}(a_j)) < 0 \tag{25}$$

This is equivalent to:

$$q^*(a_i)\texttt{Acc}_{\text{valid}}(a_i) + q^*(a_j)\texttt{Acc}_{\text{valid}}(a_j) < q^*(a_j)\texttt{Acc}_{\text{valid}}(a_i) + q^*(a_i)\texttt{Acc}_{\text{valid}}(a_j) \tag{26}$$

In this case, we can derive a new distribution $r(a)$ by only changing the probability of $q^*(a_i)$ and $q^*(a_j)$:

$$r(a) = \begin{cases} q^*(a) & a \neq a_i \text{ and } a \neq a_j \\ q^*(a_i) & a = a_j \\ q^*(a_j) & a = a_i \end{cases} \tag{27}$$

Then, we can derive that:

$$\begin{aligned}
\mathcal{R}(r)+\beta\mathcal{H}(r) =& \mathbb{E}_{a\sim r}\mathtt{Acc}_{\mathrm{valid}}(a)-\beta\mathbb{E}_{a\sim r}\log(r(a)) \\
=& \sum_{a\in\mathcal{A}} r(a)\mathtt{Acc}_{\mathrm{valid}}(a)-\beta\sum_{a\in\mathcal{A}} r(a)\log(r(a)) \\
=& \sum_{a\in\mathcal{A}-\{a_i,a_j\}} r(a)\mathtt{Acc}_{\mathrm{valid}}(a)-\beta\sum_{a\in\mathcal{A}-\{a_i,a_j\}} r(a)\log(r(a)) \\
& + r(a_i)\mathtt{Acc}_{\mathrm{valid}}(a_i)+r(a_j)\mathtt{Acc}_{\mathrm{valid}}(a_j)-\beta r(a_i)\log r(a_i)-\beta r(a_j)\log r(a_j) \\
=& \sum_{a\in\mathcal{A}-\{a_i,a_j\}} q^*(a)\mathtt{Acc}_{\mathrm{valid}}(a)-\beta\sum_{a\in\mathcal{A}-\{a_i,a_j\}} q^*(a)\log(r(a)) \\
& + q^*(a_j)\mathtt{Acc}_{\mathrm{valid}}(a_i)+q^*(a_i)\mathtt{Acc}_{\mathrm{valid}}(a_j)-\beta q^*(a_j)\log q^*(a_j)-\beta q^*(a_i)\log q^*(a_i)
\end{aligned}$$

We can apply the In-equation (26) here:

$$\begin{aligned}
\mathcal{R}(r)+\beta\mathcal{H}(r) \geq& \sum_{a\in\mathcal{A}-\{a_i,a_j\}} q^*(a)\mathtt{Acc}_{\mathrm{valid}}(a)-\beta\sum_{a\in\mathcal{A}-\{a_i,a_j\}} q^*(a)\log(r(a)) \\
& + q^*(a_i)\mathtt{Acc}_{\mathrm{valid}}(a_i)+q^*(a_j)\mathtt{Acc}_{\mathrm{valid}}(a_j)-\beta q^*(a_i)\log q^*(a_i)-\beta q^*(a_j)\log q^*(a_j) \\
=& \mathbb{E}_{a\sim q^*}\mathtt{Acc}_{\mathrm{valid}}(a)-\beta\mathbb{E}_{a\sim q^*}\log(r(a)) \\
=& \mathcal{R}(q^*)+\beta\mathcal{H}(q^*)
\end{aligned}$$

We can conclude that $r(a)$ is better than the $q^*(a)$, which is contradictory with the supposition that $q^*(a)$ is the optimal distribution that can maximize $\mathcal{R}(\theta)+\beta\mathcal{H}(\Gamma(\theta))$. Therefore, Theorem 4.2 holds. □

## C. Datasets

We give the urls to the datasets we have used with their licenses.

- **GNN Benchmark** https://www.in.tum.de/daml/gnn-benchmark/ with MIT license.
- **Open Graph Benchmak** https://ogb.stanford.edu/ with MIT license.

## D. Detailed Hyper-Parameters

*Table 6.* Detailed hyper-parameters setting.

| Dataset | #layer | lr | epoch | dropout | hidden size | sample | controller update epoch |
|---|---|---|---|---|---|---|---|
| CS | 2 | 0.005 | 100 | 0.5 | 256 | Full Batch | every 5 epochs |
| Physics | 2 | 0.005 | 100 | 0.5 | 256 | Full Batch | every 5 epochs |
| Arxiv | 3 | 0.01 | 500 | 0.5 | 256 | Full Batch | every 25 epochs |
| Products | 4 | 0.001 | 50 | 0.5 | 256 | Cluster | every 2.5 epochs |
| Papers100M | 3 | 0.001 | 25 | 0.1 | 1024 | Neighbor | every 1.25 epochs |

We report the detailed hyper-parameters for supernet training and architecture retraining in Table 6. We want to emphasize that all the hyper-parameters except for CS and Physics are set according to the published codes in OGB Leaderboard (Hu et al., 2020)[4]. The hyper-parameters for CS and Physics are set using grid search on single model GCN (Kipf & Welling, 2017).

For Products, we use cluster sub-graph sampling following ClusterGCN (Chiang et al., 2019), with the number of 15000 random partitions and a batch size of 32. For Papers100M, we use neighborhood sampling following GraphSAGE (Hamilton

[4] https://ogb.stanford.edu/docs/leader_nodeprop/

et al., 2017), with every layer sampling 12 and 100 neighbors for train and validation/test respectively. The batch size is set to 1024.

For the search strategy, we use Regularized Evolution (Real et al., 2019). We maintain a population of 50, and mutate 10 top architectures each time for 5 epochs.

## E. Runtime Environment

All the experiments, including baselines, are conducted on the following environments.

- Operating System: Ubuntu 18.04.5 LTS.
- CPU: Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz.
- GPU: NVIDIA Tesla V100-PCIE-32GB.
- Software: Python 3.7.11, PyTorch 1.10.1, PyTorch Geometric 2.0.3 (Fey & Lenssen, 2019), Deep Graph Library 0.7.2 (Wang et al., 2019), Open Graph Benchmark 1.3.2 (Hu et al., 2020).