# AUTOCOG: A UNIFIED DATA-MODAL CO-SEARCH FRAMEWORK FOR GRAPH NEURAL NETWORKS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Neural architecture search (NAS) has demonstrated success in discovering promising architectures for vision or language modeling tasks, and it has recently been introduced to searching for graph neural networks (GNNs) as well. Despite the preliminary success, we argue that for GNNs, NAS has to be customized further, due to the topological complicacy of GNN input data (graph) as well as the notorious training instability. Besides optimizing the GNN model architecture, we propose to simultaneously optimize the input graph topology, via a set of parameterized data augmentation operators. That yields **AutoCoG**, the first unified data-model co-search NAS framework for GNNs. By defining a highly flexible data-model co-search space, **AutoCoG** is gracefully formulated as a principled bi-level optimization, that can be end-to-end solved by the differentiable search methods. Experiments demonstrate that **AutoCoG** produces state-of-the-art performance at standard benchmarks including Cora, PubMed, and Citeseer, outperforming both state-of-the-art hand-crafted GNNs as well as recent GNN NAS methods. **AutoCoG** can also scale to searching deeper GCNs in larger-scale datasets. Our method consistently achieves state-of-the-art (SOTA) results on Cora, Citeseer, Pubmed, and ogbn-arxiv. Specifically, we achieve gain of up to 2.04% for Cora, 2.54% for Citeseer, 2.08% for Pubmed, and finally 0.83% for ogbn-arxiv on our benchmarks.

## 1 INTRODUCTION

Graph neural networks (GNNs) have emerged as promising tools to analyze networked data in various real-world scenarios, such as social media (Grover & Leskovec (2016)) and biochemical graph analytics (Zitnik & Leskovec (2017)). Specifically, GNNs apply recursive message passing to learn the embedding representation of each node via aggregating the representations of its neighbors and itself. Motivated by the significant success of node embedding learning, plenty of GNN variants have been explored for the diverse downstream graph analysis tasks, including GCN (Kipf & Welling (2016)), GraphSAGE (Hamilton et al. (2018)), and GCNII (Chen et al. (2020a)).

However, training GNNs is notoriously challenging, especially for noisy graphs and deep GNNs (Chen et al. (2020a)). First, since graphs abstract diverse data sources and present tremendous heterogeneity, the success of GNNs is often accompanied by extensive tuning of model architectural hyperparameters to characterize specific graph data. For example, it was reported that graph attention networks (GAT (Veličković et al. (2018b))) are sensitive to the number of attention heads, which has to be carefully searched for the citation networks and the protein-protein interaction data, respectively. Second, the real-world graphs are inevitably noisy due to the error-prone and incomplete data collection, where GNNs tend to suffer from overfitting and generalize poorly to the unseen testing data. Third, despite the potential of deep GNNs in learning the informative high-order neighborhood, the training of deep GNNs is widely known to be limited by the issues of over-smoothing, gradient vanishing, and over-squashing (Chen et al. (2020a)).

Recently, the automated graph neural architecture search (NAS), graph augmentation tricks, and deeper architectures have been independently proposed to tackle the above GNN training challenges partially. Expressly, most of the existing automated efforts are limited to neural architecture tuning, while graph augmentation is often overlooked and untouched despite often being effective to gain performance (Li & King (2020); Zhou et al. (2019b)). This is primarily because changes to the

existing graph structure can have a cascading effect on the process of information aggregation, which adds a new layer of complexity above the already complex architecture tuning problem. Another limitation of existing GNN NAS works lies in their scalability to deeper architectures. Because the options are many, the search complex and (perhaps most importantly) the training unstable, previous NAS efforts have limited themselves in searching the shallow GNNs with less than 3 layers. Thus it remains as a daunting task to comprehensively optimize the design philosophy of deep GNNs, which has been shown to deliver superior generalization performance for any given graph data.

To bridge the gaps, we propose **AutoCoG**, the first NAS framework towards **unified data-model co-search for GNNs**. Besides automatically optimizing the GNN model architecture, we propose to simultaneously optimize the input graph topology, via a set of parameterized and searchable data augmentation operators that modify graph structures. By defining the highly flexible data-model co-search space, **AutoCoG** is formulated as a principled bi-level optimization that can be end-to-end solved by the differentiable search methods. To scale up our core framework to searching deep GNN architectures, we curb an explosive search space as the number of layers increased by performing multiple searching stages with increasing depth, as inspired by (Chen et al. (2019c)). Our framework is further stabilized by (i) injecting identity mappings (Chen et al., 2020a) to combat the over-smoothing/over-squashing issues; and (ii) adding uniform noise to perturbing architecture parameters to smoothen the search landscape (Chen & Hsieh, 2021).

Together, our framework ensures a reliable way to discover powerful architectures, a stable model training environment, and state-of-the-art results. **AutoCoG** searches for and trains on deep or shallow graph neural networks to successfully achieve state-of-the-art results in Cora, Citeseer, Pubmed, and ogbn-arxiv. In summary, our three contributing novelties are:

- We propose **AutoCoG**, the first differentiable NAS framework towards unified data-model co-search for GNN. Our novel bi-level optimization formulation uniquely enables the end-to-end discovery of state-of-the-art GNN model and graph augmentation policy altogether.
- To strengthen the co-search framework, we organically integrate several techniques to directly combat issues of searching unreliability, training instability, and scalability, that have previously plagued NAS approaches for searching deeper GNNs.
- Our method consistently achieves state-of-the-art (SOTA) results on Cora, Citeseer, Pubmed, and ogbn-arxiv. Specifically, we achieve gains of up to 2.04% for Cora, 2.54% for Citeseer, 2.08% for Pubmed, and finally 0.83% for ogbn-arxiv.

## 2 Related works

**Graph neural networks.** Motivated by the state-of-the-art results of GNNs in graph analytics, there have been numerous GNN variants (Bruna et al., 2013; Hamilton et al., 2017; Xu et al., 2019; Chen et al., 2020a; Wu et al., 2019). Most of these existing approaches fit within the category of spatial GNNs. Namely, following the spatial message passing strategy, the core idea of GNNs is to learn the embedding representation of a node by aggregating the embeddings of its neighbors and node itself recursively. The previous empirical studies show that GNNs often achieve the best performance with less than 3 layers (Kipf & Welling, 2016; Veličković et al., 2018).

A key limitation of GNNs is their performances decrease significantly with the increasing of model depth. As the graph convolutional layer increases, the node representations will converge to indistinguishable vectors due to the recursive neighborhood aggregation and non-linear activation (Li et al., 2018; Oono & Suzuki, 2020), which is well recognized as over-smoothing issue (NT & Maehara, 2019; Chen et al., 2019a; Alon & Yahav, 2020; Chien et al., 2021; Huang et al., 2020). Such limitation prevents the developments of deep GNNs to model the dependencies of high-order neighbors. Recently, a series of techniques have been leveraged to tackle the over-smoothing, including skip connection (Li et al., 2019; Klicpera et al., 2018; Xu et al., 2018), random dropping (Rong et al., 2020b), and graph normalization (Zhao & Akoglu, 2019; Zhou et al., 2020). Based on these tricks, deep GNNs with more than 16 layers deliver the superior performances in the scientific citation network and biochemical graph analytics (Le et al., 2015; Liu et al., 2020).

**Graph augmentation.** Data augmentation methods has been widely applied to improve the generalization performances of deep neural networks, such as convolutional and recurrent neural net-

works Shorten & Khoshgoftaar (2019); Antoniou et al. (2017); Feng et al. (2021). They aim to craft the out-of-distribution training data to avoid overfitting with the customized augmentation policies. In the graph analytics, GNNs are prone to overfit the naturally noisy training graphs, which may miss the ground-truth nodes/edges or contain the erroneous information Zügner et al. (2018). Different from the grid-like image data, the graph augmentation is often operated on the adjacency structure or node features. The existing graph augmentations could be catagorized into the following two classes. (i) The random augmentation either drops/adds edges to modify the graph, or masks parts of the node features Rong et al. (2020b); You et al. (2020b); Feng et al. (2020). (ii) The differentiable augmentation learns to optimize the adjacency affinity matrix by minimizing the concerned task loss. Based upon the computed affinity matrix, the differentiable augmentation either continuously combines it into the original adjacency matrix Zhao et al. (2020b); Chen et al. (2020b), or samples the discrete edges to formulate new graph Chen et al. (2019b).

**Neural architecture search.** Targeting at alleviating the laborious hyperparameter tuning, NAS automates the designing of good neural architectures for any a given application. It is shockingly reported that the searched neural architectures could outperform the human-designed ones in many real-world scenarios, such as image classification Zoph & Le (2016); Zoph et al. (2018) and generation Wang & Huan (2019); Gong et al. (2019). Most of NAS frameworks apply one of the following search algorithms: reinforcement learning (RL) Pham et al. (2018); Baker et al. (2016), evolution algorithm (EA) Liu et al. (2017); Miikkulainen et al. (2019); Xie & Yuille (2017), and one-shot differentiable search Liu et al. (2018); Zela et al. (2020). There are several recent efforts to conjoin the researches of GNNs and NAS Gao et al. (2019); Zhou et al. (2019a); You et al. (2020a); Ding et al. (2020); Zhao et al. (2020a). However, all of them are limited in exploring the shallow GNNs, and fail to denoise the underlying graph to further ameliorate the model performance. In this work, we aim to simultaneously search the deep GNN models and graph structure to optimize the downstream graph analytics.

# 3 METHODOLOGY

## 3.1 UNIFIED DATA-MODAL CO-SEARCH SPACE

### 3.1.1 MODEL SEARCH SPACE: ATTENTION, ACTIVATION, AND SKIP CONNECTION

**Preliminary.** We briefly review the basic of a message-passing based graph convolution network (GCN). In a general form, its $k$-th layer could be written as:

$$\begin{aligned} h_i^{(k)} &= \mathrm{AGGR}(\{a_{ij}^{(k)} W^{(k)} x_j^{(k-1)} : j \in \mathcal{N}(i)\}) \\ x_i^{(k)} &= \sigma(\mathrm{COMB}(W^{(k)} x_i^{(k-1)}, h_i^{(k)})) \end{aligned} \tag{1}$$

$x_i^{(k)}$ denotes the node embedding of element $i$ at $k$-th layer. $W^{(k)} \in \mathbb{R}^{D \times D}$ represents the learnable layer-wise weights for all $\{x_i : i \in |V|\}$, where $|V|$ is our total number of nodes and $D$ our number of hidden features. $a_{ij}^{(k)}$ dictates the attention coefficient between $i$ and $j$ derived from some Attention function. $\mathcal{N}(i)$ denotes the neighboring nodes of node $i$ from a graph $G$. $h_i^{(k)}$ is the resulting embeddings after applying an AGGR function to aggregate a set of neighboring embeddings from the previous $k-1$ layers. In addition, function COMB incorporates information from itself with its neighboring embeddings $h_i^{(k)}$, and $\sigma$ provides the nonlinear activation.

**Defining the Model Search Space.** The design of model search space should achieve a balanced trade-off between the diversity and efficiency. Although a large search space subsumes the diverse GNN architectures to adapt to the different graph analysis tasks, it would be extremely time-consuming to explore the optimal design. In the existing search spaces of GNNs Gao et al. (2019); Zhou et al. (2019b); You et al. (2020a), they often contain the architecture components of hidden units, attention, aggregation, combination, and activation functions, as well as the skip connections. To efficiently search the outperforming shallow and deep GNNs, we compare the effectiveness of each component, and greatly shrink down the search space to focus on three key components: the Activation function, the Attention module, and the skip connections. They are generally believed

Table 1: The set of attention functions, where $||$ denotes the concatenation operation, $\bar{a}, \bar{a}_i, \bar{a}_j$ denote learn-able vectors, $W_G$ denotes the trainable matrix.

| Attention Choice | Expression Form |
|---|---|
| GCN | $\frac{1}{\sqrt{\lvert\mathcal{N}(i)\rvert\lvert\mathcal{N}(j)\rvert}}$ |
| COS | $\bar{a}(W^{(k)}x_i^{(k-1)}\|\|W^{(k)}x_j^{(k-1)})$ |
| LINEAR | $\tanh(\bar{a}_l W^{(k)}x_i^{(k-1)}\|\|W^{(k)}x_j^{(k-1)})$ |
| GERE-LINEAR | $W_G\tanh(W^{(k)}x_i^{(k-1)} + W^{(k)}x_i^{(k-1)})$ |
| GAT | $\mathrm{LeakyReLU}(\bar{a}(W^{(k)}x_i^{(k-1)}\|\|W^{(k)}x_j^{(k-1)}))$ |
| GAT-SUM | $a_{ij}^{(k)} + a_{ij}^{(k)}$ based on GAT |
| CONST | 1 |

to impact GNN's expressive capability and depth scalability (Chen et al., 2021b). We fix the Aggreation function and Combination function to be simple summation, and treat the hidden units as hyperparameter. Below we lay out our searchable design for them one-by-one:

- *Attention Search Space*: Attention mechanism has been shown by (Veličković et al., 2018b) to effectively stabilize training by placing proper neighborhood scaling with attention coefficient $a_{ij}$. We list our attention choices in Table 1.
- *Activation Search Space*: for basic activation functions, we search among these operations $\{\mathrm{ReLU}, \mathrm{Sigmoid}, \mathrm{Tanh}, \mathrm{Linear}, \mathrm{SoftPlus}, \mathrm{LeakyReLU}, \mathrm{ReLU6}, \mathrm{ELU}\}$.
- *Skip Connection Search Space*: For an $L$-Layer GCN, various skip connections can be applied to overcome the effect of over-smoothing. Previous deep GCN works (Chen et al., 2020a; Zhang et al., 2020; Chen et al., 2021b) illustrated a significant correlation between the type of skip-connections to the performance. We include three skip-connection types:
  - Dense connection: $x_i^{(k)} = C(\{x_i^{(l)} : 0 \leq l \leq k\})$
  - Initial connection: $x_i^{(k)} = (1-\eta) * x_i^{(k)} + \eta * x_i^{(0)}$
  - Jumping connection: $x_i^{(L)} = C(\{x_i^{(l)} : 0 \leq l \leq L\}$. Note that jumping connection is a special case of dense connection applying on the last layer.

  Our search could choose from each of the three types. Further, $\eta$ is a learnable hyperparameter, and $C$ is a set of searchable combination functions $\{\mathrm{mean}, \mathrm{max}, \mathrm{concat}\}$.

#### 3.1.2 Data Search Space: Graph Augmentations

A given graph $G(V, E)$ can be expressed in the form of an adjacent matrix $A \in \mathbb{R}^{|V|\times|V|}$, where $V$ is the set of vertices and $E$ is the set of edges. Graph augmentation is the transform of $G$ such that $\hat{G} = A \bigodot m$, where $m \in \{0,1\}^{|V|\times|V|}$ is a binary matrix derived from some augmentation policy or operator. Correspondingly, we could rewrite the aggregation step in Eqn (1) as $H^{(k)} = AX^{(k-1)}W^{(k)}$, where $H^{(k)} = \{h_i^{(k)} : 0 \leq i \leq |V|\}$ and $X^{(k-1)} = \{x_i^{(k-1)} : 0 \leq i \leq |V|\}$.

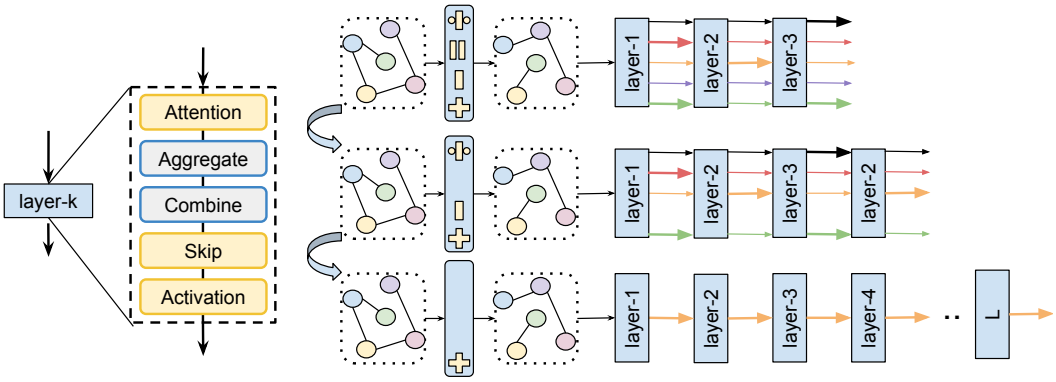
Despite being relatively overlooked by traditional GNN literature, a number of prior works (Srivastava et al., 2014; Zou et al., 2019; Rong et al., 2020a; Chen et al., 2021a; Huang et al., 2021) have found that graph augmentations, i.e., by (randomly) perturbing a certain number of edges or nodes from the input graph, can helps decelerate both the over-fitting and over-smoothing issues in training deep GNNs. Formally, besides searching the GNN model search architecture, we define our jointly searchable space of graph augmentation operators below:

- DropEdge (Rong et al., 2020a): $H^{(k)} = (A \bigodot m)X^{(k-1)}W^{(k)}$, where $m \in \{0,1\}^{|V|\times|V|}$ is a binary random matrix, and each element $m_{ij} \sim B(1,p)$ is drawn from Bernoulli distribution with probability of $p$.
- AddEdge: $H^{(k)} = (A||m)X^{(k-1)}W^{(k)}$, where $m \in \{0,1\}^{|V|\times|V|}$ is a binary matrix, $||$ is the OR function, the new total edges is $|E| + p * |E|$, and each new edge is randomly sampled.

- DropNode (Huang et al. (2021)): $H^{(k)} = (QA\text{diag}(m)Q^T)X^{(k-1)}W^{(k)}$ where $m \in \{0,1\}^{|V|}$ is a binary random matrix, each element $m_j \sim B(1,p)$ is drawn from Bernoulli distribution with $p$ probability. $Q \in 0, 1^{s\times|V|}$ our node selection matrix, for our selected node set $\{i_i : 0 \leq i \leq s\}$, where $Q_{k,m} = 1$ if $m = i_k$.
- Identity: $H^{(k)} = AX^{(k-1)}W^{(k)}$

Since most of our data policies are sampled from a Bernoulli distribution with $p$ probability, $p$ is just as important as the data policies themselves, and hence will be searched as a hyperparameter as well. Practically, for any policy except Identity, we search $p$ from $\{0.1, 0.15, 0.2, 0.25, 0.3\}$.

Figure 1: An overview of **AutoCoG** data-model co-search architecture and pipeline. On the left, we deconstruct a layer into its components denoted in Eqn. (1). Yellow indicates searchable components in the model search space, while light-blue functions are fixed. The data (graph augmentation) search space is illustrated as a series of mathematical signs: (+) edge add, (-) edge permutation, (÷) node drop and (=) identity. The search is conducted in a differentiable and depth-progressive fashion.




### 3.2 Optimization Formulation and Algorithm

#### 3.2.1 A Principled Bi-Level Optimization Formulation

For the sake of conciseness, we use $\alpha$ as the *model space architecture parameters*, and meanwhile denote $\beta$ as the set of *data space architecture parameters*, consisting of the choice graph augmentation policy parameters and $p$. Both $\alpha$ and $\beta$ are categorical variables, and together they characterize a joint search outcome. Now, let us denote our $\mathbf{L}_{\text{obj}}$ the objective loss function given $\alpha$ and $\beta$. Given $\beta$, we can write our augmented graph $\hat{G}$ as $\hat{A} = A \bigodot m_\beta$ where $m_\beta \in \{0,1\}^{|V|\times|V|}$. Similarly, given $\alpha$, we denote $\hat{W} = W \bigodot m_\alpha$ as the pruned sub-model from the supernet derived from $\alpha$ description, where $\hat{W}, m_\alpha \in \mathbf{R}^{L\times D\times D}$. Then, let $Z$ represents our output vector for a hypothetical 2-layer **AutoCoG**:

$$Z = \text{Softmax}((\hat{A}\sigma(\hat{A}X\hat{W}^{(0)})\hat{W}^{(1)})) \tag{2}$$

Thus the objective loss function $\mathbf{L}_{\text{obj}}$ for a transductive semi-supervised node classification tasks is formally denoted as:

$$\mathbf{L}_{\text{obj}}(\hat{G}, \hat{W}) = -\frac{1}{|Y_{\text{label}}|}\sum_{y_i \in Y_{\text{label}}} y_i \log(z_i) \tag{3}$$

Extending from (Dong & Yang, 2019), we formulate our data-model co-search as a **joint bi-level optimization**, to solve $\alpha, \beta$ concurrently with the weights $W$ and data space parameters:

$$\begin{aligned} &\min_{\alpha,\beta} \mathbf{L}_{\text{obj}}^{\text{valid}}(\hat{W}, \hat{G}, \beta, \alpha) \\ &\text{s.t.} \quad \hat{W}, \hat{G} = \arg\min_{W,G} \mathbf{L}_{\text{obj}}^{\text{train}}(W, G, \beta, \alpha) \end{aligned} \tag{4}$$

Note that $\alpha, \beta$ are optimized using the objective loss function on the validation set, while $\hat{W}, \hat{G}$ are optimized under training set. We adopt the same hard-*Gumbel-softmax* trick (Jang et al., 2017) to differentially optimize all categorical variables.

#### 3.2.2 SCALING AND STABILIZING THE SEARCH

The bi-level optimization (4) can be solved by differential search methods, and we adopt the GDAS approach in (Dong & Yang, 2019) by default. However, when exploring GCN deep architectures and larger graphs, the data/model search spaces grow exponentially with the layer depth/graph size, and they can be entangled to cause even more serious scalability challenge. That is further amplified by the training difficulty and instability of deep GNNs (Chen et al., 2021b). Indeed, we observe that naively applying GDAS is prone to over-smoothening and search collapse, only yielding very poor architectures when searching for more than three layers. Besides, it is not uncommon for the derived graph and model to have considerable performance variations across repeated experiments, due the stochastic initialization and training.

We investigate and incorporate the following three "tricks" into our differential search process. They are found to contribute remarkably to the scalability, stability and consistency of **AutoCoG**.

**Progressive search space.** We follow the idea proposed by Chen et al. (2019c) (also illustrated in Figure 1), to divide search into $N$ progressive stages, with each consecutive stage having a larger or equal number of layers than those previously. At each stage, we greedily remove the least selected options (by taking the mean of Soft-Max across $L$ layers and removing the option with the smallest value) from the data or model space, and pass on the shrunk co-search space to the next stage.

**Identity mapping.** To relieve the over-smoothing and over-fitting issues, the Identity Mapping was first proposed by Chen et al. (2020a) to fashion deep GCN. It is formally denoted as:

$$X^{(k)} = \sigma(AX^{(k-1)}(B_k W^{(k)} + (1 - B_k)I)) \tag{5}$$

Where $B_k = \log(\frac{\gamma}{k} + 1)$ and $\gamma$ is a positive hype-parameter.

**Architecture parameter perturbation.** We add small perturbations $\delta \sim U_{[-\epsilon,\epsilon]}$ to both model space and data space architecture parameters. Such an addition was first proposed by Chen & Hsieh (2021) to more effectively sample the search space while preventing saddle points from hindering architecture progress. We find it to be a helpful stabilizer too. We scale $\epsilon$ linearly between 0.01 to 3. During evaluation, no perturbation is added. With this trick, we partially rewrite Eqn (4) as:

$$\hat{W}, \hat{G} = \arg\min_{W,G} \mathbf{L}_{\text{obj}}^{\text{train}}(W, G, \beta + \delta, \alpha + \delta) \tag{6}$$

Eventually after search, we use the $\alpha$ and $\beta$ parameters that resulted in the highest validation accuracy to derive the architecture, which will be re-trained. For re-training, we perform 100 separate runs, each with a maximum of 1000 epochs, and report the average accuracy plus standard deviation.

## 4 EXPERIMENTS

### 4.1 DATASETS AND EXPERIMENTAL SETTINGS.

**Datasets.** Following the previous efforts, we evaluate **AutoCoG** on four popular node classification datasets (i) Cora, (ii) Citeseer, (iii) Pubmed (Sen et al. (2008)) and (iv) ogbn-arxiv (ogbn-arxiv) (Hu et al. (2020)). The maximum diameters of the strongly connected components on Cora, Citeseer, Pubmed, ogbn-arxiv are 19, 28, 18, and 23, respectively, where the high-order neighborhood modeling is a challenging task to boost the node classification results. They are widely adopted to compare the effectiveness of NAS approaches, graph augmentation methods, shallow and deep GNNs. In these citation graphs, nodes and edges correspond to the scientific documents and their citation relationships, respectively. The node embedding features are the bag-of-words representation of documents (Cora, Citeseer or Pubmed) or the average embeddings of words in title and abstract (ogbn-arxiv). Table 2 details the statistic of each dataset.

**Experimental settings.** We extend the default settings used in Chen et al. (2020a), while adding some of our own. For example, we use the Adam optimizer (Kingma & Ba (2017)) for both model and architecture weights with a learning rate of $0.01$, and $0.001$ respectively. $L_2$ regularization for the model's weight is set to be $0.0005$ as standard for Cora/Citeseer/Pubmed, and zero for ogbn-arxiv. We set dropout rates to be 0.6 for Cora and Citeseer; 0.5 for Pubmed; and 0.1 for ogbn-arxiv. For Identity-Mapping, $\gamma$ is chosen to be 0.5 for Cora/Pubmed/ogbn-arxiv, and 0.6 for Citeseer. Following best practices, we set the hidden dimension to be 256 for Citeseer/Arvix, and 64 for Pubmed/Cora. Number of head is one for all. In addition, we set the $\eta$, ratio between initial connection and current features, to be 0.1. For P-DARTS, we set $N$ to be seven with an increment of two layers per stage. Finally, we allow our search and train to reach a maximum of 1000 epochs, while setting our patience to be 400, and 200 respectively.

Table 2: Dataset Statistic

| dataset | classes | nodes | edges | features |
|---|---|---|---|---|
| Cora | 7 | 2,708 | 5,429 | 1,433 |
| Citeseer | 6 | 3,327 | 4,732 | 3,703 |
| Pubmed | 3 | 19,717 | 44,338 | 500 |
| ogbn-arxiv | 40 | 169,343 | 1,666,243 | 128 |

### 4.2 RESULTS

**Baselines.** To better realize **AutoCoG**'s potential, we compare it with extensive baselines at 2/16/32 layers configuration:

- Standard GNNs: (i) GCN (Kipf & Welling (2016)), (ii) SGC (Wu et al. (2019)), (iii) GAT (Veličković et al. (2018a)). We also try to apply graph augmentations on GCN and SGC.
- Deeper GNNs: (i)GCNII (Chen et al. (2020a)), (ii) JKNet (Xu et al. (2018)), (iii) DAGNN (Liu et al. (2020)), (iv) APPNP (Klicpera et al. (2018)), (v) GPRGNN (Chien et al. (2021)).
- NAS-based GNNs:(i) AGNN (Zhou et al. (2019b)), (ii) GraphNAS (Gao et al. (2019)).

Note that we put (*) next to **AutoCoG**'s 32-layers results for it is actually a 25 layers model. This is done to avoid OOM error; nevertheless, it still achieves SOTA results.

**Results.** We report our results under Table (3). **Compare to standard GNNs**, we are clearly a cut above with gains over the next best of 2.04%/ 9.31%/ 16.84% for 2/ 16/ 32 layers for Cora, and similar gains repeated in Citeseer (-1.08%/ 2.54%/ 12.22%), Pubmed (2.08%/ 11.64%/ 14.01%) and ogbn-arxiv (1.02%/ 6.55%/ 30.73%). With the most notable exceptions being 2-layer Citeseer, where we perform worst than both GCN and SGC by 0.13% and 1.08% respectively. **Compare to deeper GNNs**, we achieve better results across the board too, with gains over the next best approach being (1.60%/ 0.6%/ 0.0%) for Cora, (-0.44%/ 0.52%/ 0.9%) Citeseer, (0.98%/ 0.52%/ 0.03%) Pubmed and (0.83%/ 0.08%/ 0.09%) ogbn-arxiv. Notable exceptions being a tie with GCNN for 32-layer Cora and being worst than APPNP for 2-layer Citeseer. **Against other NAS-based approaches,** we achieve gains of (0.53%/ 53.39%/ 53.39%) for Cora, (-2.57%/ 46.39%/ 46.65%) for Citeseer, (0.74%/ 41.01%/ 37.44%) for Pubmed. With notable exception being 2-layer Citeseer, where we scored on average worst than previous NAS works by $2.4\%$ and where Auto-GNN achieved the best performance overall for shallow network.

### 4.3 ABLATION STUDIES

#### 4.3.1 A CASE FOR CO-ADAPTATION

To demonstrate the necessity of co-adaption, we compare results of random-search, model/data only search, simple augmentation of standard GNNs, and our co-adaptation on the Cora dataset. For Data-only search, we fix the attention to be GCN, disable any skip connections, and set our activation function as ReLU. We report our findings of random-search and model/data-only search in Table (4), while our co-adaption and simple augmentation results can be find in Table (3).

Table 3: Semi-supervised transductive results between **AutoCoG** and other SOTA approaches. We report the average best accuracy across 100 runs and the standard deviation. We denote (0.2) as probability $p$ of graph augmentation policy. Accuracy is in percent (%), and best results are in bold.

| Dataset | Type | Method | layers 2 | 16 | 32 |
|---|---|---|---|---|---|
| Cora | Standard | GCN | 80.68±0.13 | 28.56±2.77 | 29.36±2.76 |
| | | SGC | 79.31±0.37 | 75.98±1.06 | 68.45±3.10 |
| | | GAT | 82.09±0.56 | 15.67±8.32 | 13.55±3.74 |
| | Graph-augmentation | GCN+DropNode(0.2) | 77.10±1.04 | 27.61±4.34 | 27.65±5.07 |
| | | GCN+DropEdge(0.2) | 79.16±0.73 | 28.00±3.36 | 27.87±3.04 |
| | | SGC+DropNode(0.2) | 77.89±0.23 | 75.22±0.22 | 69.51±0.40 |
| | | SGC+DropEdge(0.2) | 78.16±0.24 | 70.65±0.80 | 44.00±0.90 |
| | Deeper | JKNet | 79.06±0.11 | 72.91±3.94 | 73.23±3.59 |
| | | GCNII | 82.19±0.77 | 84.69±0.51 | **85.29±0.47** |
| | | DAGNN | 80.30±0.78 | 84.14±0.59 | 83.39±0.59 |
| | | APPNP | 82.06±0.46 | 83.64±0.48 | 83.68±0.48 |
| | | GPRGNN | 82.53±0.49 | 83.69±0.55 | 83.13±0.60 |
| | NAS | Auto-GNN | 83.60±0.30 | 22.34±6.55 | 31.90±0.00 |
| | | GraphNAS | 82.70±0.40 | 31.90±0.00 | 31.90±0.00 |
| | | **AutoCoG** | **84.13±0.58** | **85.29±0.25** | **85.29± 0.48** |
| Citeseer | Standard | GCN | 71.36±0.18 | 23.19±1.47 | 23.03±1.13 |
| | | SGC | 72.31±0.38 | 71.03±1.18 | 61.92±3.48 |
| | | GAT | 71.07±0.49 | 21.44±4.50 | 8.16±2.63 |
| | Graph-augmentation | GCN+DropNode(0.2) | 69.38±0.89 | 21.83±3.07 | 22.18±3.06 |
| | | GCN+DropEdge(0.2) | 70.26±0.70 | 22.92±1.95 | 22.92±2.12 |
| | | SGC+DropNode(0.2) | 71.87±0.27 | 72.50±0.20 | 70.60±0.11 |
| | | SGC+DropEdge(0.2) | 71.94±0.32 | 69.43±0.57 | 45.13±0.93 |
| | Deeper | JKNet | 66.98±1.82 | 54.33±7.74 | 50.68±8.73 |
| | | GCNII | 67.81±0.89 | 72.97±0.71 | 73.24±0.78 |
| | | DAGNN | 18.22±3.48 | 73.05±0.62 | 72.59±0.54 |
| | | APPNP | 71.67±0.78 | 72.13±0.53 | 72.13±0.59 |
| | | GPRGNN | 70.49±0.95 | 71.39±0.73 | 71.01±0.79 |
| | NAS | Auto-GNN | **73.80±0.70** | 22.95±4.57 | 26.78±0.85 |
| | | GraphNAS | 73.50±1.00 | 27.18±1.64 | 27.49±0.62 |
| | | **AutoCoG** | 71.23±0.75 | **73.57±0.34** | **74.14±0.48** |
| Pubmed | Standard | GCN | 77.39±0.98 | 40.18±1.18 | 39.88±2.54 |
| | | SGC | 78.06±0.31 | 69.18±0.58 | 66.61±0.56 |
| | | GAT | 78.36±0.73 | 24.45±10.42 | 18.00±0.00 |
| | Graph-augmentation | GCN+DropNode(0.2) | 77.39±0.98 | 40.18±1.18 | 39.88±2.54 |
| | | GCN+DropEdge(0.2) | 78.26±0.32 | 68.39±0.26 | 52.08±0.79 |
| | | SGC+DropNode(0.2) | 77.63±0.32 | 70.28±0.21 | 68.16±0.33 |
| | | SGC+DropEdge(0.2) | 79.16±0.73 | 28.00±3.36 | 27.87±3.04 |
| | Deeper | JKNet | 77.24±0.92 | 64.37±8.80 | 63.77±9.21 |
| | | GCNII | 78.05±1.53 | 80.03±0.50 | 79.91±0.27 |
| | | DAGNN | 77.74±0.57 | 80.32±0.38 | 80.58±0.51 |
| | | APPNP | 79.46±0.47 | 80.30±0.30 | 80.24±0.33 |
| | | GPRGNN | 78.73±0.63 | 78.78±1.02 | 78.46±1.03 |
| | NAS | Auto-GNN | 79.70±0.40 | 38.76±5.66 | 41.26±1.59 |
| | | GraphNAS | 78.80±0.50 | 39.81±1.65 | 43.17±1.54 |
| | | **AutoCoG** | **80.44±0.21** | **80.82±0.17** | **80.61±0.20** |
| ogbn-arxiv | Standard | GCN | 69.53±0.19 | 66.14±0.73 | 41.96±9.01 |
| | | SGC | 61.98±0.08 | 41.58±0.27 | 34.22±0.04 |
| | | GAT | 71.05±0.18 | 40.59±15.17 | 22.09±4.03 |
| | Graph-augmentation | GCN+DropNode(0.2) | 66.67±0.16 | 67.17±0.47 | 43.81±9.62 |
| | | GCN+DropEdge(0.2) | 68.67±0.17 | 66.38±0.60 | 45.74±5.65 |
| | | SGC+DropNode(0.2) | 61.21±0.08 | 40.52±0.11 | 34.64±0.05 |
| | | SGC+DropEdge(0.2) | 62.06±0.05 | 41.03±0.23 | 33.61±0.06 |
| | Deeper | JKNet | 63.73±0.38 | 66.41±0.56 | 66.31±0.63 |
| | | GCNII | 71.24±0.17 | 72.61±0.29 | 72.60±0.25 |
| | | DAGNN | 67.65±0.52 | 71.82±0.28 | 71.46±0.27 |
| | | APPNP | 65.31±0.23 | 66.95±0.24 | 66.94±0.26 |
| | | GPRGNN | 69.31±0.09 | 70.30±0.15 | 70.18±0.16 |
| | NAS | Auto-GNN | OOM | OOM | OOM |
| | | GraphNAS | OOM | OOM | OOM |
| | | **AutoCoG** | **72.07±0.12** | **72.69±0.10** | **72.69±0.06*** |

Herein, we see a strong performance deficit between Data-only and **AutoCoG** at (0.72%/ 53.26%/ 53.2%) for 2/16/32 layers, respectively. While model-only search fairs somewhat better with only a

deficit of (0.61%/ 1.06%/ 0.38%). In addition, we also observe an interesting performance difference between Data-only search and GCN-DropNorm/GCN-DropEdge at $p = 0.2$ with gains of (4.25%/ 4.03%/ 4.22%).

These observations validate three aspects of our methodology: (i) we get far better results than random, which indicates that our search space is non-trivial and our methodology effective. (ii) we perform better together than as individual components, validating our co-adaptation aim (iii) We show that naive application of graph augmentation has detrimental effects on performance, while searched policy enjoys far better accuracy. Thus the experiment empirically demonstrates the validity of our proposed methodology.

Table 4: To understand the important of co-search, we isolate each components of our framework to study their individual effectiveness. Data-only search is based on the GCN framework.

| Dataset | Method | layers | | |
|---|---|---|---|---|
| | | 2 | 16 | 32 |
| Cora | Random | 63.53±0.25 | 27.51±0.64 | 30.69±2.19 |
| | Data-only | 83.41±0.66 | 32.03±0.19 | 32.09±0.15 |
| | Model-only | 83.52±0.44 | 84.23±0.20 | 84.91±0.22 |
| | **AutoCoG** | **84.13±0.58** | **85.29±0.25** | **85.29± 0.48** |

### 4.3.2 Effectiveness of stabilizers

Similar to how we study the effectiveness of co-adaptation, we compare **AutoCoG**'s results without some or all of our stabilizing "tricks" and discuss their overall effectiveness. We report the our findings under table 5. Without the small noise added to the architecture parameters, we observe significant impact to the performance of 2-layer configuration while having minimal effects on 16/32 configurations.

Since **AutoCoG** searches directly on two layers without using progressive search, it is more vulnerable to minor variation in initialization. On the other hand, 16/32-layer configurations can mitigate this instability by relying on P-DART. Meanwhile, we observe a small performance deficit of only (.17%/ .34%/ .41%), when identity mapping is missing. Nevertheless, when both features are missing, we can observe a more significant performance gap of (7%/ 1%/ 0.7%) for 2, 16, and 32, respectively. The combined deficit to performance when both features are missing validate our usage in order to to achieve SOTA performance.

Table 5: To study the effectiveness of our stabilizing "tricks", we test each of them individually to study their effectiveness

| Dataset | Method | layers | | |
|---|---|---|---|---|
| | | 2 | 16 | 32 |
| Cora | without architecture perturbation | 82.50±0.56 | 84.33±0.58 | 85.11±0.27 |
| | without identity mapping | 83.96±0.24 | 84.95±0.34 | 84.87±0.33 |
| | without both | 77.88±1.66 | 84.42±0.27 | 84.60±0.23 |
| | **AutoCoG** | **84.13±0.58** | **85.29±0.25** | **85.29± 0.48** |

## 5 Conclusion

In this paper, we present **AutoCoG** the first NAS framework towards **unified data-model co-search for GNNs**. We design a highly flexible data-model co-search space aiming to formulated as a principled bi-level optimization, that can be end-to-end solved by the differentiable search methods. Our results convincingly demonstrates the benefit of data-graph co-search for both deep and shallow graph neural networks. We show **AutoCoG** to be a reliable way to discover powerful architectures, a stable model training environment, and state-of-the-art results. For future works, (i) we plan to expand our search space to encompassed all aspects of GNN's building blocks. (ii) we want consider more advance techniques in graph-augmentation, beyond the current random base approaches.

**Reproducibility Statement.** As part of the reproducibility effort, we have painstakingly recorded all derived architectures, random seeds, and trained weights. We shall disclose within the supplementary the details of our derived architectures at all configurations for all datasets. Upon acceptance, we will publicly release our code and all associated trained models weights used to achieve the reported accuracy.

## REFERENCES


Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv*, 2016.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv*, 2013.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *arXiv preprint arXiv:1909.03211*, 2019a.

Deli Chen, Xiaoqian Liu, Yankai Lin, Peng Li, Jie Zhou, Qi Su, and Xu Sun. Highwaygraph: Modelling long-distance node relations for improving general graph neural network, 2019b.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks, 2020a.

Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In *International Conference on Machine Learning*, pp. 1695–1706. PMLR, 2021a.

Tianlong Chen, Kaixiong Zhou, Keyu Duan, Wenqing Zheng, Peihao Wang, Xia Hu, and Zhangyang Wang. Bag of tricks for training deeper graph neural networks: A comprehensive benchmark study. *arXiv preprint arXiv:2108.10521*, 2021b.

Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization, 2021.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1294–1303, 2019c.

Yu Chen, Lingfei Wu, and Mohammed Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *Advances in Neural Information Processing Systems*, 33, 2020b.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations. https://openreview. net/forum*, 2021.

Yuhui Ding, Quanming Yao, and Tong Zhang. Propagation model search for graph neural networks. *arXiv preprint arXiv:2010.03250*, 2020.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours, 2019.

Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. A survey of data augmentation approaches for nlp. *arXiv preprint arXiv:2105.03075*, 2021.

Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. *Advances in Neural Information Processing Systems*, 33, 2020.

Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graphnas: Graph neural architecture search with reinforcement learning, 2019.

Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3224–3234, 2019.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeuIPS*, pp. 1024–1034, 2017.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Wenbing Huang, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. Tackling over-smoothing for general graph convolutional networks. *arXiv e-prints*, pp. arXiv–2008, 2020.

Wenbing Huang, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. Tackling over-smoothing for general graph convolutional networks, 2021.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9267–9276, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Yaoman Li and Irwin King. Autograph: Automated graph neural network. In *International Conference on Neural Information Processing*, pp. 189–201. Springer, 2020.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv*, 2017.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv*, 2018.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 338–348, 2020.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pp. 293–312. Elsevier, 2019.

Hoang NT and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv*, 2018.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations. https://openreview. net/forum*, 2020b.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, Sep. 2008. doi: 10.1609/aimag.v29i3.2157. URL https://ojs.aaai.org/index.php/aimagazine/article/view/2157.

Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018a.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018b.

Hanchao Wang and Jun Huan. Agan: Towards automated design of generative adversarial networks. *arXiv*, 2019.

Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr. au2, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks, 2019.

Lingxi Xie and Alan Yuille. Genetic cnn. In *ICCV*, pp. 1379–1388, 2017.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33, 2020a.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33, 2020b.

Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. *arXiv preprint arXiv:2001.10422*, 2020.

Hongwei Zhang, Tijin Yan, Zenjun Xie, Yuanqing Xia, and Yuan Zhang. Revisiting graph convolutional network on semi-supervised node classification from an optimization perspective, 2020.

Huan Zhao, Lanning Wei, and Quanming Yao. Simplifying architecture search for graph neural network. In *International Conference on Information and Knowledge Management*, 2020a. URL http://ceur-ws.org/Vol-2699/paper08.pdf.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. *arXiv preprint arXiv:2006.06830*, 2020b.

Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. Auto-gnn: Neural architecture search of graph neural networks. *arXiv preprint arXiv:1909.03184*, 2019a.

Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. Auto-gnn: Neural architecture search of graph neural networks, 2019b.

Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. Towards deeper graph neural networks with differentiable group normalization. *Advances in Neural Information Processing Systems*, 33, 2020.

Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, Jul 2017. ISSN 1460-2059. doi: 10.1093/bioinformatics/btx252. URL http://dx.doi.org/10.1093/bioinformatics/btx252.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv*, 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, pp. 8697–8710, 2018.

Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks, 2019.

Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2847–2856, 2018.

# A APPENDIX

In this section, we outline the derived architectures from **AutoCoG** at all configurations for all datasets. Below you can find our experimental settings; next to it are the three tables, each outlines the architecture configurations for two layers, 16 layers, and 32 layers.

**Experimental settings.** We use the Adam optimizer for both model and architecture weights with a learning rate of $0.01$, and $0.001$ respectively. $L_2$ regularization for the model's weight is set to be $0.0005$ as standard for Cora/Citeseer/Pubmed, and zero for ogbn-arxiv. We set dropout rates to be 0.6 for Cora and Citeseer; 0.5 for Pubmed; and 0.1 for ogbn-arxiv. For Identity-Mapping, $\gamma$ is chosen to be 0.5 for Cora/Pubmed/ogbn-arxiv, and 0.6 for Citeseer. Following best practices, we set the hidden dimension to be 256 for Citeseer/Arvix, and 64 for Pubmed/Cora. Number of head is one for all. In addition, we set the $\eta$, ratio between initial connection and current features, to be 0.1. For P-DARTS, we set $N$ to be seven with an increment of two layers per stage. Finally, we allow our search and train to reach a maximum of 1000 epochs, while setting our patience to be 400, and 200 respectively.

Table 6: architectures derived for our 2 layers settings. the left most setting are for layer 1, while the right most settings are for layer 2.

| dataset | data policy ($p$) | attention | activation | skip |
|---|---|---|---|---|
| Cora | DropNode (0.15) | const, gat | tanh, tanh | None, None |
| Citeseer | DropNode (0.10) | cos, gcn | relu6, leaky relu | None, None |
| Pubmed | Identity | gcn, gcn | tanh, tanh | initial skip, initial skip |
| ogbn-arxiv | DropNode (0.15) | gcn, gcn | tanh, tanh | inital skip, inital skip |

Table 7: architectures derived for our 16 layers settings. The same setting is repeated across the depth of our model

| dataset | data policy ($p$) | attention | activation | skip |
|---|---|---|---|---|
| Cora | DropNode (0.2) | gcn | linear | initial skip |
| Citeseer | Identity | gcn | tanh | initial skip |
| Pubmed | AddEdge (0.1) | gcn | tanh | initial skip |
| ogbn-arxiv | DropNode (0.15) | gcn | tanh | inital skip |

Table 8: architectures derived for our 32 layers settings. The same setting is repeated across the depth of our model

| dataset | data policy ($p$) | attention | activation | skip |
|---|---|---|---|---|
| Cora | AddEdge (0.15) | gcn | elu | initial skip |
| Citeseer | AddEdge (0.10) | gcn | tanh | initial skip |
| Pubmed | AddEdge (0.10) | gcn | tanh | initial skip |
| ogbn-arxiv | DropNode (0.10) | gcn | tanh | inital skip |