Graph Neural Networks with Node-wise Architecture

Zhen Wang
Alibaba Group
jones.wz@alibaba-inc.com

Zhewei Wei
Renmin University of China
zhewei@ruc.edu.cn

Yaliang Li*
Alibaba Group
yaliang.li@alibaba-inc.com

Weirui Kuang
Alibaba Group
weirui.kwr@alibaba-inc.com

Bolin Ding
Alibaba Group
bolin.ding@alibaba-inc.com

ABSTRACT

Recently, Neural Architecture Search (NAS) for GNN has received increasing popularity as it can seek an optimal architecture for a given new graph. However, the optimal architecture is applied to all the instances (i.e., nodes, in the context of graph) equally, which might be insufficient to handle the diverse local patterns ingrained in a graph, as shown in this paper and some very recent studies. Thus, we argue the necessity of node-wise architecture search for GNN. Nevertheless, node-wise architecture cannot be realized by trivially applying NAS methods node by node due to the scalability issue and the need for determining test nodes' architectures. To tackle these challenges, we propose a framework wherein the parametric controllers decide the GNN architecture for each node based on its local patterns. We instantiate our framework with depth, aggregator and resolution controllers, and then elaborate on learning the backbone GNN model and the controllers to encourage their cooperation. Empirically, we justify the effects of node-wise architecture through the performance improvements introduced by the three controllers, respectively. Moreover, our proposed framework significantly outperforms state-of-the-art methods on five of the ten real-world datasets, where the diversity of these datasets has hindered any graph convolution-based method to lead on them simultaneously. This result further confirms that node-wise architecture can help GNNs become versatile models.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Machine learning algorithms**.

KEYWORDS

Graph Neural Networks; Neural Architecture Search; Dynamic Neural Networks

ACM Reference Format:
Zhen Wang, Zhewei Wei, Yaliang Li, Weirui Kuang, Bolin Ding. 2022. Graph Neural Networks with Node-wise Architecture. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539387

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539387

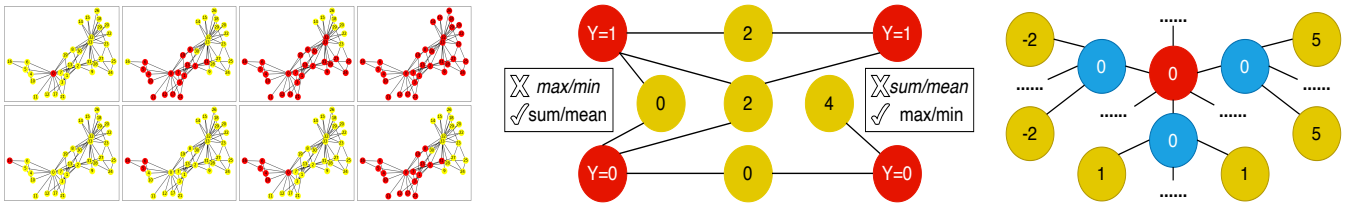
1 INTRODUCTION

In recent years, Graph Neural Networks (GNNs) [4, 29] have been proposed and applied to solve various tasks on ubiquitous graph data, including social networks [15], citation networks [16], and biological networks [38]. When applying GNN to a new graph, Neural Architecture Search (NAS) for GNN [5, 7, 18, 28, 34–37] is often conducted to seek a suitable GNN architecture for handling that graph, e.g., choosing the depth of GNN to be 3 from the candidate depths $\{2, 3, \cdots\}$, choosing the mean pooling from the candidate pooling operations {min, max, mean}, and so on.

Existing works in this line follow the convention of NAS to apply the searched optimal architecture to all the instances (i.e., nodes) equally. However, for different nodes in the same graph, their local patterns, including both the topological structures and the node attributes in their neighborhoods, are usually very diverse, making applying the same architecture to all nodes unsuitable. As a piece of evidence, researchers have recently observed the different levels of local assortativity exhibited in each of several real-world graphs [24], which lead to unsatisfactory performances of several representative GNNs [21]. Therefore, we argue that GNN with node-wise architecture is much in demand.

To be specific, we present three examples in Fig. 1 that justify the necessity of using node-wise architecture from three different aspects of architecture. (1) Different nodes may need different *depths* for the GNN. Comparing the two rows of Fig. 1a, the message of a densely connected node propagates much faster than that of a node with rare connections. This phenomenon has been analyzed as that nodes with a larger degree are more quickly to produce over-smoothed node embeddings along with the iterations of graph convolution operations [1, 23]. Node-wise depth has been studied in recent works [14, 30] to allow nodes with different local structures to have different depths while avoiding the over-smoothing issue. These works support our idea of node-wise architecture from the aspect of depth. (2) Different nodes may need different *aggregators*. In Fig. 1b, with the assumed class labels and in-coming messages, the two target nodes on the left-hand side can be successfully distinguished by a mean/sum pooling, while the two nodes on the right-hand side require a max/min pooling. These two pairs have been used by PNA [3] to motivate the usage of a mixture of aggregators for a GNN. In our case, we emphasize the necessity of selecting the appropriate aggregator in a node-wise manner. (3) We

*Corresponding author.



(a) Depth (b) Aggregator (c) Resolution

Figure 1: Examples that motivate node-wise architecture for GNN. (a) Each row denotes a message propagation process, where red color indicates the message has been propagated to the node. (b) Nodes with red color are target nodes to predict. The numbers shown in yellow color nodes denote their messages to be propagated towards the target nodes. (c) Blue color nodes and yellow color nodes are 1-hop and 2-hop neighbors of the red color node, where the numbers denotes the node attributes.

propose a novel notion, the *resolution* of a GNN layer, as how many neighbors are sampled for aggregating their messages. Sampling is necessary for training GNN models on large graphs, where different resolutions often lead to different computation graphs and thus architectures of the applied GNN. In practice, the widely-adopted neighbor sampler [8] uses the same pre-specified resolutions of the GNN layers for all nodes. However, when the local patterns of a node are the case of the red node shown in Fig. 1c, sampling more 1-hop neighbors would be better than sampling more 2-hop neighbors in the sense of reducing the variance of estimation. Thus, this node prefers a high resolution for the second GNN layer while being insensitive to the resolution for the first layer.

Motivated by these observations, we study how to search for the optimal GNN architecture in a node-wise manner. To this end, a straightforward extension of existing NAS methods will increase the size of search space linearly w.r.t. the number of nodes, which makes it intractable on large-scale graphs. Moreover, such an extension searches the suitable architectures only for the nodes that are accessible during training, and thus it cannot generate the suitable architectures for the test nodes under the inductive setting.

To tackle these challenges, we propose a framework wherein there is a parametric controller for each aspect of architecture, e.g., an aggregator controller, to decide which kind of aggregator should be applied. To determine the architecture configuration for a node at a specific layer, the controller first encodes the node's local patterns into a context embedding and then takes choices from the search space based on it. For example, when we assume that features of the node at previous layers are sufficient for determining its desired aggregator at the current layer, we can feed the embeddings of the node at previous layers into the aggregator controller. Intuitively, the backbone GNN model depends on the controllers to predict the suitable architectures, while the controllers make predictions for each node based on the node's local patterns captured by the backbone model. Thus we design a learning method to promote the cooperation between the backbone model and the controllers.

Since our context-aware controllers have a fixed number of parameters independent of the number of nodes and can generalize to unseen nodes, the proposed framework can achieve node-wise architecture for a GNN model even on a large-scale graph under both transductive and inductive settings. It is worth noting that although we instantiate the proposed framework with depth, aggregator, and resolution controllers, controllers designated for other aspects of GNN architecture can be easily included in our framework.

We compare our proposed framework with state-of-the-art methods on ten real-world graph datasets. Our method achieves the best performances on half of them, where no or at most one baseline method can reach our 95% confidence interval. Moreover, the datasets we outperform include homophilic and heterophilic graphs, where no existing graph convolution-based method can lead simultaneously. Meantime, the node-wise resolution is shown to improve the performance of a GNN on a large-scale graph. Then we show that the controllers can appropriately correlate the suitable node-wise depth, aggregator, and resolution with each node's local patterns, which explains how node-wise architecture can improve the performance of GNNs. The payment for realizing such node-wise architecture is also empirically evaluated from the perspective of sample efficiency and running time.

2 PRELIMINARIES

We first introduce the notations used in this paper, and give a brief summary of GNN models and NAS for GNN. Let $G = (\mathcal{V}, \mathcal{E})$ denote a graph with node attributes $\mathbf{x}_v$ for each node $v \in \mathcal{V}$. Without loss of generality, we consider undirected graphs in this paper, and thus the neighborhood of a node v can be denoted by $\mathcal{N}(v) = \{u|(u, v) \in \mathcal{E}\}$. In this paper, we focus on node-level tasks (e.g., node classification) where each node v is associated with a label $y_v \in \mathcal{Y}$. Our goal is to learn a GNN from the labeled nodes to predict the unlabeled ones.

GNN. Existing GNN models, spatial-based [8, 31] or spectral-based [4, 9, 12], are often described and implemented in the message passing paradigm, where the representation $\mathbf{h}_v^{(l)}$ of node v at the l-th layer is recursively calculated by aggregating the messages propagated from its neighbors. This calculation can be formulated as

$$\mathbf{h}_v^{(l)} = \sigma(\text{Aggr}_{u \in \mathcal{N}(v) \cup \{v\}}(\phi^{(l)}(\mathbf{h}_u^{(l-1)}))), \mathbf{h}_v^{(0)} = \mathbf{x}_v, \quad (1)$$

where $l = 1, \ldots, L$, $\sigma(\cdot)$ denotes an activation function (e.g., ReLU), $\phi^{(l)}(\cdot)$ denotes any differentiable function such as an MLP, and $\text{Aggr}(\cdot)$ denotes a permutation-invariant aggregator such as the mean pooling. In general, we can use $\mathbf{h}_v^{(L)}$ as the final node representation $\mathbf{z}_v$ for predicting y_v.

NAS for GNN. Works in this line has studied some aspects of the architecture for a GNN, e.g., the intra-layer design needs to determine the $\text{Aggr}(\cdot)$ and the inter-layer design needs to seek for an optimal depth L [34]. Each aspect is associated with its search space, e.g., the $\text{Aggr}(\cdot)$ is allowed to take choice from $\mathcal{O} = \{\text{mean, add, max, min}\}$. Differentiable NAS [19, 36], one of the most widely adopted NAS

approaches, often models each aspect by a random variable, e.g., O with possible outcomes $\mathcal{O}$, and parameterizes $\Pr(O)$ by a $|\mathcal{O}|$-dimensional vector $\boldsymbol{\phi}$ in the form—$\Pr(O = o; \boldsymbol{\phi}) = \frac{\exp(\boldsymbol{\phi}_o)}{\sum_{o' \in \mathcal{O}} \exp(\boldsymbol{\phi}_{o'})}$. Conventionally, $\boldsymbol{\phi}$ is called architecture parameter to be distinguished from the model parameter $\boldsymbol{\theta}$ of the backbone GNN model. Then the search procedure corresponds to optimizing the architecture parameter $\boldsymbol{\phi}$.

3 GNNS WITH NODE-WISE ARCHITECTURE

Generally, existing works in the line of NAS for GNN search for an optimal architecture and apply it to all the nodes equally. Suppose the searched architecture corresponds to a GNN that applies mean pooling in its first layer and max pooling in its second layer. In Fig. 2, when we apply this searched architecture to the graph, the GNN models applied to the three nodes (A, E, and G) will result in the computation graphs shown in the "ordinary GNN" part.

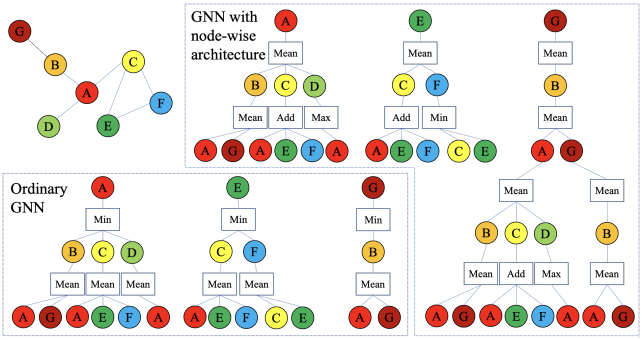


Figure 2: An example to illustrate the difference between node-wise architecture and using the same architecture.

However, as discussed in Sec 1, it might be unsatisfactory to apply the same architecture for handling all the nodes, and thus GNN with node-wise architecture is needed. Before introducing how to realize GNN with node-wise architecture, we first present an example in Fig. 2 to show its difference against an ordinary GNN. For depth, the GNN applied to node G has four layers while the GNNs applied to other nodes have a depth of two, where the difference might come from their different node degrees. For aggregator, the GNN applied to node B uses a mean pooling at its first layer while that applied to node F uses a min pooling.

3.1 Context-aware Controller

For the differentiable NAS discussed in Sec. 2, suppose there is an aspect of the architecture to be determined in each of the L layers, then the distributions $\Pr(O^{(l)}), l = 1, \ldots, L$ are parameterized by the architecture parameters with a total dimension of $L \times |\mathcal{O}|$. If we attempt to realize node-wise architecture via a straightforward extension of such NAS method, there would be a dedicated random variable $O_v^{(l)}$ for each node $v \in V$. Thus, the total dimension of the required architecture parameters will increase along with the number of nodes $|V|$ linearly, which is unaffordable on large graphs. Moreover, in an inductive setting, the test nodes are inaccessible until the test phase, where the architecture parameters for the test nodes cannot be estimated in advance.

To achieve GNN with node-wise architecture on large graphs, we propose a framework that utilizes parametric controllers to predict the suitable architectures for the backbone GNN model. The controller makes predictions for each node based on its context that can reflect its local patterns. Thus different nodes are allowed to have different GNN architectures. To determine a specific aspect of the GNN architecture, we characterize the node-wise distribution $\Pr(O_v^{(l)})$ by $g(\cdot)$, which will encode its input into a context embedding to reflect the local patterns of the node v at the stage of the l-th layer and output a distribution over $\mathcal{O}$. We are allowed to consider different inputs for controllers responsible for different aspects of GNN architecture, with the principle that the inputs should provide sufficient evidence for determining the suitable architecture.

In our framework, any aspect of GNN architecture can be handled by simply adding a corresponding controller. We exemplify the proposed context-aware controller from the aspects of depth, aggregator, and resolution as follows.

Depth controller. We present two different designs for the depth controller $g^{(\mathrm{d})}(\cdot)$. In the first design, given the maximal allowed depth L, we can define the search space as $\mathcal{O} = \{0, \ldots, L\}$, the node-wise distribution $\Pr(O_v) = g^{(\mathrm{d})}(\{\mathbf{h}_v^{(l)}, l = 0, \ldots, L\})$, and the final node representation $\mathbf{z}_v$ as follow:

$$\mathbf{z}_v = \sum_{o=0}^{L} \Pr(O_v = o)\mathbf{h}_v^{(o)}. \tag{2}$$

In the other design, we let the controller to make a choice from the search space $\mathcal{O} = \{0, 1\}$ at each layer, where "1" means to terminate at that layer. Then we define the node-wise and layer-wise distribution by $\Pr(O_v^{(l)}) = g^{(d)}(\{\mathbf{h}_u^{(l)} | u \in \mathcal{N}(v) \cup \{v\}\})$ and calculate the final node representation of a node v as follow:

$$\mathbf{z}_v = \sum_{l=0}^{L} (\Pr(O_v^{(l)} = 1) \prod_{k=0}^{l-1} (1 - \Pr(O_v^{(k)} = 1)))\mathbf{h}_v^{(l)}, \tag{3}$$

where the products express the probability of being terminated at the l-th layer but not any of the previous layers.

Aggregator controller. In addition to the dimension-wise pooling operations, we also include a special "self_msg" operation which receives the message of the target node itself while ignoring any incoming message. Then the search space of our aggregator controller can be expressed as $\mathcal{O} = \{\text{max}, \text{min}, \text{add}, \text{mean}, \text{self\_msg}\}$. With the aggregator controller $g^{(\mathrm{a})}(\cdot)$, the first step in each message passing iteration is to predict the aggregator to be applied, according to $\Pr(O_v^{(l)}) = g^{(\mathrm{a})}(\{\mathbf{h}_u^{(l-1)} | u \in \mathcal{N}(v) \cup \{v\}\})$. Given the predicted distribution $\Pr(O_v^{(l)})$ over $\mathcal{O}$, the message passing procedure defined in Eq. (1) becomes:

$$\mathbf{h}_v^{(l)} = \sigma(\sum_{o \in \mathcal{O}} \Pr(O_v^{(l)} = o) o_{u \in \mathcal{N}(v) \cup \{v\}}(\phi(\mathbf{h}_u^{(l-1)}))). \tag{4}$$

In most practical cases, $\phi^{(l)}(\cdot)$ defined in Eq. (1) is implemented by an MLP. Since the different choices of the aggregator often lead to drastically different statistics of their outputs [37], it would be unstable in optimizing a $\phi^{(l)}(\cdot)$ shared among the candidate aggregators. Thus, we allow each aggregator $o \in \mathcal{O}$ to have a dedicated transformation $\phi_o^{(l)}(\cdot)$.

Resolution controller. Sampling is indispensable when we train a GNN model on large graphs because an entire three-hop neighborhood can often fail to fit into the GPU memory, not to mention a larger neighborhood. In this paper, we consider one of the most widely adopted samplers—neighbor sampler [8], where a fixed number of nodes are sampled in each hop. We define a GNN layer's resolution as the number of nodes sampled in the corresponding hop and regard resolution as one aspect of GNN architecture. Then the search space of the resolution controller consists of several concrete resolution configurations, e.g., $\mathcal{O} = \{15\text{-}10\text{-}5, 14\text{-}10\text{-}7, 16\text{-}8\text{-}5\}$, where "15-10-5", means sampling 15, 10, and 5 neighbors in the 3, 2, and 1-hop, respectively. It is worth noting that, in most cases, sampling is conducted only for training but evaluation, where the estimated node embedding $\hat{\mathbf{h}}_v^{(l)}$ and the exact node embedding $\mathbf{h}_v^{(l)}$ are calculated based on a sampled or an entire neighborhood of v, respectively. In each time of evaluation, we infer the exact node embeddings for all the nodes and maintain their final node representations, i.e., $\mathbf{z}_v, v \in \mathcal{V}$. We assume $\mathbf{z}_v$ is informative for determining the suitable resolution for node v and choose the resolution for it according to $o \sim \Pr(O_v) = g^{(\mathrm{r})}(\mathbf{h}_v^{(0)}, \mathbf{z}_v, y_v)$.

3.2 Modelling and Optimization

In the proposed framework, controllers are not restricted to any particular functional form. We discuss some choices to show what patterns in a node's context should be recognized by the controllers for making their decisions. For $g^{(\mathrm{d})}(\{\mathbf{h}_v^{(l)}, l = 1, \ldots, L\})$, we can feed $(\mathbf{h}_v^{(1)}, \ldots, \mathbf{h}_v^{(L)})$ into a LSTM [10] and produce the distribution based on its last hidden state. In this way, the order matters, which reflects the intuition that the depth controller observes how the node embedding changes along the iterations of message passing and then attends to the suitable layer. For $g^{(\mathrm{d})}(\{\mathbf{h}_u^{(l)} | u \in \mathcal{N}(v) \cup \{v\}\})$, the intuition is to compare the embedding of target node $\mathbf{h}_v^{(l)}$ with the embeddings of its neighbors, so that the controller can determine to terminate at the current layer when it finds that the embeddings have been similar to some extent. Thus, a simple choice is to define $\Pr(O_v^{(l)} = 1) = \frac{1}{1+\exp\{a\}}$ with $a = b + \frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)}(\mathbf{h}_v^{(l)})^{\mathrm{T}}\mathbf{W}\mathbf{h}_u^{(l)}$, where b and $\mathbf{W}$ are trainable parameters of $g^{(\mathrm{d})}(\cdot)$. For $g^{(\mathrm{a})}(\{\mathbf{h}_u^{(l-1)} | u \in \mathcal{N}(v) \cup \{v\}\})$, we aim to let the controller choose suitable aggregator based on some basic statistics of the neighbors' embeddings. Thus, we can parameterize $g^{(a)}(\cdot)$ as an MLP fed with the concatenation of $\mathbf{h}_v^{(l-1)}$ and the max, min, add, and mean values of $\{\mathbf{h}_u^{(l-1)}, u \in \mathcal{N}(v)\}$. As for $g_\phi^{(\mathrm{r})}(\mathbf{h}_v^{(0)}, \mathbf{z}_v, y_v)$, we simply feed an MLP with the concatenation of $\mathbf{h}_v^{(0)}$, $\mathbf{z}_v$, and y_v.

Let us denote the parameters of the backbone GNN model by $\boldsymbol{\theta}$, the parameters of the controllers by $\boldsymbol{\phi} = (\boldsymbol{\phi}_{\mathrm{d}}, \boldsymbol{\phi}_{\mathrm{a}}, \boldsymbol{\phi}_{\mathrm{r}})$ where the subscripts imply the corresponding controllers. According to Eq. (2), Eq. (3), and Eq. (4)), we notice that the final node representation $\mathbf{z}_v$ depends on both $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. Intuitively, the controllers make predictions for each node based on the node embeddings calculated by the backbone GNN model, while the backbone GNN model depends on the controllers to predict the suitable architectures. Thus, to encourage their cooperation, we learn $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ jointly. On the one hand, only $\boldsymbol{\phi}_{\mathrm{d}}$ and $\boldsymbol{\phi}_{\mathrm{a}}$ directly participate in the forward propagation to produce $\mathbf{z}_v$ and thus can be optimized in a differentiable manner, e.g., making gradient descent like DARTS [19] or making exponentiated gradient descent like GAEA [17]. On the other hand, we regard the resolution controller $g_{\boldsymbol{\phi}_{\mathrm{r}}}(\cdot)$ as a parametric policy with action space (i.e., candidate resolutions) $\mathcal{O}_{\mathrm{r}}$, from which we sample the resolutions. Since we cannot directly calculate the gradients of $\mathbf{z}_v$ w.r.t. the sampled resolutions, we adopt policy gradient method [25] to optimize it. The goal of $g_{\boldsymbol{\phi}_{\mathrm{r}}}(\cdot)$ is to select a suitable resolution configuration for each node v, such that the estimated node embedding $\hat{\mathbf{h}}_v^{(l)}$ can better approximate the exact one $\mathbf{h}_v^{(l)}$. To this end, we design the reward function to be: $R(v, o) = -\|\mathbf{z}_v - \hat{\mathbf{z}}_v\|_2^2, v \in \mathcal{V}, o \in \mathcal{O}_{\mathrm{r}}$. The pseudo-code for learning both $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ can be found in Algorithm 1.

Algorithm 1 Learning a GNN with node-wise architecture.

Input: Graph $G = (\mathcal{V}, \mathcal{E})$ with splits $\mathcal{V}_{\text{train}}$ and $\mathcal{V}_{\text{valid}}$, Graphs sampler S, learning rate α, and #epochs T.
Output: Learned parameters $\boldsymbol{\phi} = (\boldsymbol{\phi}_{\mathrm{d}}, \boldsymbol{\phi}_{\mathrm{a}}, \boldsymbol{\phi}_{\mathrm{r}})$ and $\boldsymbol{\theta}$.

Randomly initialize $\boldsymbol{\phi}^{(0)}$ and $\boldsymbol{\theta}^{(0)}$;
for $t = 0, 1, \ldots, T-1$ **do**
  Infer $\mathbf{z}_v$ for $v \in \mathcal{V}$ by $\boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}_{\mathrm{d}}^{(t)}, \boldsymbol{\phi}_{\mathrm{a}}^{(t)}$
  $o_v \sim \Pr_{\boldsymbol{\phi}_{\mathrm{r}}}(O_v) = g_{\boldsymbol{\phi}_{\mathrm{r}}^{(t)}}(\mathbf{h}_v^{(0)}, \mathbf{z}_v, y_v)$ for $v \in \mathcal{V}_{\text{valid}}$;
  $G_{\text{valid}} \sim S(G, v, o_v), v \in \mathcal{V}_{\text{valid}}$; // graph sampling
  Infer $\hat{\mathbf{z}}_v$ for $v \in \mathcal{V}_{\text{valid}}$ from G_{valid} by $\boldsymbol{\theta}^{(t)}, \boldsymbol{\phi}_{\mathrm{d}}^{(t)}, \boldsymbol{\phi}_{\mathrm{a}}^{(t)}$;
  $\boldsymbol{\phi}_{\mathrm{r}}^{(t+1)} \leftarrow \boldsymbol{\phi}_{\mathrm{r}}^{(t)} + \alpha\nabla_{\boldsymbol{\phi}_{\mathrm{r}}} \frac{1}{|\mathcal{V}_{\text{valid}}|}\sum_{v\in\mathcal{V}_{\text{valid}}} R(v, o_v) \log \Pr_{\boldsymbol{\phi}_{\mathrm{r}}}(O_v = o_v)$; // policy gradients
  $\boldsymbol{\phi}_{\mathrm{d}}^{(t+1)} \leftarrow \boldsymbol{\phi}_{\mathrm{d}}^{(t)} - \alpha\nabla_{\boldsymbol{\phi}_{\mathrm{d}}}\mathcal{L}_{\text{valid}}(\boldsymbol{\phi}_{\mathrm{d}}^{(t)}, \boldsymbol{\phi}_{\mathrm{a}}^{(t)}, \boldsymbol{\theta}^{(t)})$ and $\boldsymbol{\phi}_{\mathrm{a}}^{(t+1)} \leftarrow \boldsymbol{\phi}_{\mathrm{a}}^{(t)} - \alpha\nabla_{\boldsymbol{\phi}_{\mathrm{a}}}\mathcal{L}_{\text{valid}}(\boldsymbol{\phi}_{\mathrm{d}}^{(t)}, \boldsymbol{\phi}_{\mathrm{a}}^{(t)}, \boldsymbol{\theta}^{(t)})$; // gradient descent
  $o_v \sim g_{\boldsymbol{\phi}_{\mathrm{r}}^{(t+1)}}(\mathbf{h}_v^{(0)}, \mathbf{z}_v, y_v)$ for $v \in \mathcal{V}_{\text{train}}$;
  $G_{\text{train}} \sim S(G, v, o_v)$ for $v \in \mathcal{V}_{\text{train}}$; // graph sampling
  $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \alpha\nabla_{\boldsymbol{\theta}}\mathcal{L}_{\text{train}}(\boldsymbol{\psi}_{\mathrm{d}}^{(t+1)}, \boldsymbol{\psi}_{\mathrm{a}}^{(t+1)}, \boldsymbol{\theta}^{(t)})$; // gradient descent
end for
return $\boldsymbol{\phi}^{(T)}$ and $\boldsymbol{\theta}^{(T)}$.

3.3 Discussion

Our proposed framework provides a unified view for achieving GNN with node-wise architecture, which enables: (1) controllers responsible for different aspects to be fed with respective information as if it is suitable for determining the corresponding aspect; (2) new aspects of the architecture to be added; (3) controllers to be optimized jointly with the backbone GNN model, no matter they are differentiable or not.

Connections to related works. There are several recent works that our framework can express. Ala-GCN [30] terminates the iteration of message passing when an indicator for over-smoothing is active, which is roughly the similarity between $\mathbf{h}_v$ and $\{\mathbf{h}_u | u \in \mathcal{N}(v)\}$ and can be regarded as a non-parametric version of Eq. (4). IterGNN [26] determines the depth of GNN on-the-fly similarly as Eq. (3), but not in a node-wise manner. Policy-GNN [14] also uses a policy to determine each node's depth, where policy gradient

method is utilized; Considering RL's notorious sample complexity and the massive variance of policy gradients, we prefer to update the controllers by gradient descent unless they are not differentiable. PNA [3] improves the expressiveness by considering a mixture of different aggregators, where the mixing coefficients are not flexibly determined based on the local patterns of each node similar to our aggregator controller. More methods that determine a specific aspect of each node's architecture include JKNet [32], GAT [27], and GeniePath [20]. When our framework is restricted to only one controller of their corresponding aspect, our framework degenerates to these methods.

Significance. Generalizing a single aspect of architecture to more than one is nontrivial, as the enlarged search space poses difficulties in learning the controllers. To the best of our knowledge, we first attempt to consider more than one aspect and provide rigorous sample complexity studies about this generalization (see Sec. 4.3). Besides, we introduce the concept of resolution for GNN architecture and design a corresponding controller, which has not been considered before but is helpful for handling large graphs (see Sec. 4.1.2). More importantly, NW-GNN is motivated by handling the diverse local patterns ingrained in a graph. In contrast, prior works, e.g., Ala-GCN and Policy-GNN, focus on utilizing node-wise depth to alleviate the over-smoothing issue. To confirm our motivation, we will empirically show in Sec. 4.1.1 the advantages of our framework in performing well on the graphs where diverse levels of local assortativity exist. GNN architectures with fixed spectral property (e.g., low-pass) cannot perform well on such graphs, while GPR-GNN achieves satisfactory performances by explicitly adjusting the graph spectrum. In this sense, our empirical results provide an exciting motivation for node-wise architecture—the potential of promoting an ordinary GNN architecture to express a more broad scope of graph filters.

4 EXPERIMENTS

In this section, we first compare the performance of our proposed framework with state-of-the-art methods on several real-world datasets. Then we empirically justify the effects of depth, aggregator, and resolution controllers, respectively. To better understand both the benefits and burdens of node-wise architecture, we conduct experiments to study the sample efficiency and running time. To keep our notations terse, we use NW-GNN to denote the instantiation of our proposed framework.

4.1 Main Results on Real-world Datasets

4.1.1 Study on Both Homophilic and Heterophilic Graphs.

Datasets. We follow [2] to adopt ten node classification datasets that are diverse enough for making a fair and comprehensive comparison. We defer their details to Appendix (refer to Table 6 for their statistics). Their sizes (i.e., number of nodes) vary in a broad range. Moreover, according to their levels of homophily, i.e., $\mathcal{H}(G) = \frac{1}{|V|}\sum_{v\in V}\frac{|\{u|u\in\mathcal{N}(v)\wedge y_u=y_v\}|}{|\mathcal{N}(v)|}$ [22], the first five datasets are homophilic, while the last five ones are heterophilic.

Settings. We follow the "dense split" setting of [2], where the node set of each graph is randomly partitioned into train/valid/test sets with a ratio of 60%/20%/20%. We produce ten random splits on each dataset and conduct hyper-parameter optimization (HPO) for each method, where the optimal hyper-parameter configuration is determined w.r.t. the mean accuracy over the valid sets. We report the mean accuracy on the test sets for comparison.

Methods. We instantiate our proposed framework by incorporating a vanilla backbone GNN model with depth and aggregator controllers. We defer the study about resolution controller to Sec. 4.1.2 because the scales of the datasets considered here allow full-batch training. Then we categorize our baselines into three classes:

(1) Manually designed architectures: Conventionally, we adopt MLP and the widely adopted GNN architectures including ChebyNet [4], GCN [12], GraphSAGE [8], GIN [31], APPNP [13], and GPR-GNN [2] as the baselines to be compared.

(2) NAS without node-wise architecture: Then a NAS-related baseline naturally comes up, which uses the same backbone and search spaces as that of the proposed method NW-GNN. Specifically, this baseline exhaustively enumerates the possible depths and aggregators and applies the searched optimal architecture to all the nodes equally. We call this baseline *NAS** for short, which serves as NW-GNN's optimal counterpart under the standard NAS setting.

(3) Single-aspect Node-wise architectures: We consider GAT [27], JKNet [32], PNA [3], and Policy-GNN [14] which can be regarded as adaptively determining the incoming neighbors, depth, aggregator, and depth, respectively.

We implement NW-GNN and NAS* with the open-source GNN package—GraphGym [34], and the other baselines with their available open-source implementations. More details about implementation can be found in Appendix B.

Results and Analysis. We present the results in Table 1, where the bold letters imply the best result on each dataset. Overall, NW-GNN achieves best performances on half of the ten datasets, where none or at most one baseline can reach its 95% confidence interval.

(1) NW-GNN is versatile: The datasets on which NW-GNN outperforms the baselines include both homophilic and heterophilic graphs. No graph convolution-based methods (i.e., all except for MLP, APPNP, and GPR-GNN) can lead on them simultaneously. For example, GCN consistently performs well on homophilic graphs, but its performances on heterophilic graphs fall behind the leading ones by significant margins.

(2) A single searched architecture is insufficient: Compared to GCN, NAS* can use different architectures on different graphs and leads to relatively balanced performances on these two genres of graphs. However, it entirely fails to handle the heterophilic graph Actor. In contrast, NW-GNN exhibits competitive performances on all these graphs. As NAS* takes a single choice of the optimal depth and aggregator, it essentially serves as ablation of node-wise architecture. Meanwhile, some recent studies have shown that achieving satisfactory performances on these considered heterophilic graphs depends more on the capacity of addressing the diverse levels of assortativity ingrained in the graph. Therefore, we attribute NW-GNN's versatility to its capability of adjusting depth and aggregator in a node-wise manner.

(3) Node-wise architecture really helps for handling diverse local patterns: Let us take Chameleon and Squirrel as examples, where the local assortativity of node span in a broad range (see Fig. 7 in Appendix). The larger local assortativity is, the more homophilic a node is. As illustrated in Fig. 3, NW-GNN consistently outperforms those traditional graph convolution-based models on all the levels

Table 1: Results on real-world datasets: Mean accuracy (%) with its 95% confidence interval.

	Cora	CiteSeer	PubMed	Computers	Photo	Chameleon	Actor	Squirrel	Texas	Cornell
MLP	76.99±1.40	75.11±1.24	86.80±0.40	84.58±0.60	88.90±0.52	46.93±1.54	39.22±0.66	30.62±0.80	90.49±4.05	**90.65±4.09**
GCN	87.22±0.74	79.86±0.78	88.80±0.53	88.57±0.54	93.13±0.27	60.53±1.35	33.98±0.76	46.78±0.89	77.38±4.18	76.07±4.80
ChebyNet	86.96±0.72	79.29±1.14	88.92±0.36	89.21±0.37	94.87±0.22	61.31±1.36	39.46±0.75	42.32±0.93	86.06±2.14	82.45±2.61
GraphSAGE	87.63±1.56	79.78±0.82	90.29±0.61	90.53±0.31	94.60±0.25	65.51±1.36	40.63±0.85	48.99±0.65	79.03±1.20	71.41±1.23
GIN	83.74±0.90	75.95±1.20	88.38±0.40	57.18±0.21	69.03±22.38	40.18±13.74	32.81±1.05	28.70±2.77	79.67±3.78	78.36±1.56
APPNP	88.10±0.73	79.58±0.70	88.35±0.23	86.38±0.39	93.43±0.32	53.76±1.44	39.55±1.01	36.40±1.50	88.36±2.59	90.00±2.71
GPR-GNN	**88.48±0.51**	79.49±1.15	90.90±0.65	88.70±0.45	93.95±0.44	67.26±1.49	40.74±0.53	52.31±1.09	**91.48±2.02**	89.67±2.65
NAS*	87.09±1.08	80.27±0.89	88.32±0.56	89.78±0.18	93.30±0.26	67.66±1.25	34.58±0.54	55.79±1.52	89.74±3.65	90.59±3.95
GAT	88.03±0.62	80.70±0.60	88.13±0.59	90.28±0.29	93.60±0.36	66.48±1.02	35.98±0.23	53.31±1.16	78.87±0.86	76.00±1.01
JKNet	87.08±0.89	77.86±0.75	87.68±0.42	86.91±1.14	92.55±0.57	64.20±1.92	33.64±0.56	44.72±0.48	75.53±1.16	66.73±1.73
PNA	83.71±1.14	74.76±1.19	83.38±0.75	89.88±0.60	93.13±0.27	**72.32±1.28**	30.29±0.76	55.20±2.60	78.52±4.58	67.05±5.56
Policy-GNN	87.88±1.98	**82.92±8.32**	87.76±0.70	OOT	89.88±2.32	68.07±1.51	35.36±0.39	55.76±2.18	77.55±12.12	77.61±1.76
NW-GNN	88.03±0.78	79.36±0.89	**91.22±0.38**	**91.27±0.14**	**95.12±0.23**	69.06±0.93	**44.48±0.69**	**56.64±0.48**	81.80±3.59	84.26±3.47

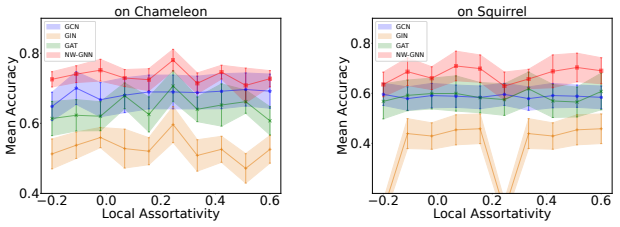


Figure 3: Performances v.s. local assortativity on two heterophilic graphs.

of local assortativity. Considering that GPR-GNN adjusts the graph spectrums directly to handle both genres of graphs well, our architecture controllers have the potential to extend the expressiveness of an ordinary GNN, acting a more broad scope of graph filters.

(4) Our controllers can choose the "ground-truth" architecture: All GNN-based methods are outperformed by MLP on Cornell, and NW-GNN outperforms most graph convolution-based methods on Cornell, where the topological information might not be helpful for node classification. Our advantage is rooted in the correct decisions made by our aggregator controller, where the "self_msg" operator is preferred. In this way, NW-GNN can degenerate into an MLP. As for the other graph convolution-based methods, although they either add self-loop or consider a residual path, the weights for in-coming and self messages cannot be adjusted as effectively as our aggregator controller does. The assertion that topological information is not helpful is supported by the learned spectral coefficients of GPR-GNN. Among the polynomial coefficients learned by GPR-GNN, only the coefficient of the zero-order term is a large positive number; those corresponding to higher-order terms are close to zero.

To better understand the behavior of the controllers, it would be helpful to show the relationships between the local patterns and the searched node-wise architectures on these datasets. However, in our controllers, node features and topological structures together determine the predicted architectures. Thus, identifying such relationships is similar to explaining the behavior of GNNs, where typical algorithms such as GNNExplainer [33] cannot provide satisfactory explanations on these adopted datasets. Therefore, we have to study such relationships on synthetic datasets in Sec. 4.2.

4.1.2 *Study on Large-scale Graph.*

Protocol. To evaluate the proposed resolution controller, we conduct experiment on a real-world large-scale dataset ogbn-products [11]. This dataset provides a node classification task on a graph with 2,449,029 nodes and 61,859,140 edges. Its scale makes full-batch training of GNN impracticible, and thus graph sampling becomes necessary. Hence, we adopt GraphSAGE as the backbone and train the model with neighbor sampler [8]. Specifically, we aim to train a three-layer GraphSAGE model, which, at each step, requires the 3-hop neighborhood of each target node in the current mini-batch to be sampled. Without resolution controller, the sampler uses the default resolution "15-10-5", that is, to sample 15, 15, and 5 neighbors at the 1th, 2nd, and 3rd hop, respectively. When coupled with our resolution controller, its search space consists of "15-10-5", "14-12-5", "16-10-2", and "15-9-7". The controller is responsible for determining a resolution from this space in each step, based on the context (i.e., local patterns). We update the controller according to what we have elaborated in Sec. 3.2. In this experiment, we focus on comparing GraphSAGE without the resolution controller (denoted by "w/o") to that with the resolution controller (denoted by "w/"). We run each setting for ten times and report its mean test accuracy.

Table 2: GraphSAGE w/o or w/ the resolution controller on ogbn-products: Mean test accuracy (%) with Std. deviation.

Metric	w/o	w/
Mean test accuracy	78.29±0.16	**78.92±0.50**

Results and Analysis. We show the experimental results in Table 2, where the performance of the baseline (i.e., "w/o") is directly copied from the leaderboards of OGB. GraphSAGE with our resolution controller (i.e., "w/") outperforms that without a controller, where one std. below the mean of the former is still higher than the mean of the latter. This comparison confirms the advantages of adaptive node-wise resolutions in training GNNs with a graph sampler, which also suggests the effectiveness of our proposed resolution controller.

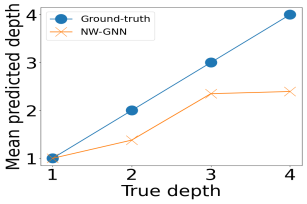


Figure 4: Correlatioin between predicted and ground-truth depths.

4.2 Justification of the Controllers' Effects

We have shown the benefits of node-wise architecture via the advantages of NW-GNN on real-world datasets, where the impacts of the introduced controllers entangle together. In the following experiments, we study the effects of depth, aggregator, and resolution controllers on three synthetic datasets, each of which calls for node-wise depth, aggregator, and resolution, respectively. More details about the construction of these synthetic datasets and the implementations can be found in Appendix B.2.

4.2.1 Effect of Node-wise Depth.

For each node in our randomly generated graph, we label it by counting the number of nodes in its k-hop neighborhood, where $k \sim \text{Uniform}(\{1, 2, 3, 4\})$. We consider three different settings about node attributes: (1) "w/o feat.": no node attribute; (2) "hard feat.": the ground-truth depth (i.e., k) for each node can be inferred from its local information; and (3) "easy feat.": the ground-truth depth for each node is given in its attributes. We compare NW-GNN to both GCN and GPR-GNN on this node regression task, where mean absolute error (MAE) on the test set is reported.

Table 3: Results of the node-wise depth study: Mean absolute error (MAE) with its 95% confidence interval.

Method		w/o feat.	hard feat.	easy feat.
GCN	$L = 2$	**1.95±0.00**	1.88±0.01	1.46±0.06
	$L = 3$	1.96±0.00	1.95±0.01	1.61±0.03
	$L = 4$	**1.95±0.00**	1.98±0.01	1.67±0.03
GPR-GNN	$\alpha = 0.1$	2.22±0.00	2.21±0.00	1.83±0.02
	$\alpha = 0.5$	2.08±0.01	2.09±0.01	1.79±0.02
	$\alpha = 0.7$	2.11±0.03	2.08±0.04	1.81±0.05
NW-GNN		1.97±0.01	**1.67±0.09**	**0.38±0.08**

The results are presented in Table 3 where NW-GNN is comparable w.r.t. GCN when there is no node attribute while surpasses all the baselines by remarkable margins under the other two settings. We attribute these advantages to our capability of using node-wise depths, so that our method can make predictions by directly approximating the generation process of ground-truth labels. To validate this capability, we conduct a case study about the relationships between our controller's predicted depth and the ground-truth depth and present the results in Fig. 4. Although the predictions made by our controller are imperfect, the average predicted depth positively correlates with the ground-truth depth. Meanwhile, under the "easy feat." setting, the MAE upon test nodes that have a correctly predicted depth is 0.15 while that upon test nodes with wrong depths is 0.53. These observations confirm the importance of using node-wise depth and the advantage of our depth controller over any fixed depth, which justify our conjecture.

4.2.2 Effect of Node-wise Aggregator.

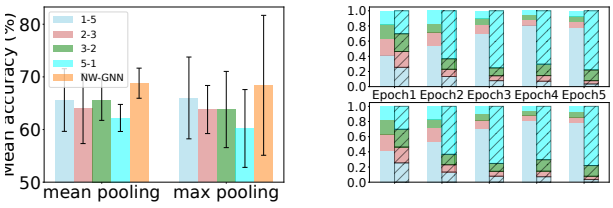
For each node in our graph, we randomly generate a numeric attribute by sampling from Uniform($[-1.0, 1.1]$) and two 5-dimensional one-hot attributes. Then we label each node by propagating their numeric attribute for two iterations, where the aggregators used are indicated by its two one-hot attributes, with the choices from {max, min, add, mean, self_msg}. We consider three different settings about node attributes: (1) "w/o feat.": Only the numeric attribute; (2) "hard feat.": The one-hot attribute corresponding to the second iteration of message passing is given; and (3) "easy feat.": The two one-hot attributes are given. We compare NW-GNN to Message Passing Neural Networks (MPNN) [6], GCN, and GPR-GNN on this node regression task, where the mean absolute error (MAE) on the test set is adopted as the measure of performance. We present the results in Table 4, where NW-GNN surpasses all the baselines under all the settings. To validate that the advantages of NW-GNN come from its capacity of choosing aggregators in a node-wise manner, we conduct a case study for NW-GNN under the "easy feat." setting. Specifically, we regard predicting the ground-truth aggregator as a 5-class classification task, and the accuracy of our controller is 54%. Meanwhile, the MAE on test nodes whose aggregators are correctly predicted is 0.78 while that on incorrectly predicted nodes is 2.02. These results confirm that NW-GNN can achieve node-wise aggregator for the backbone GNN model.

Table 4: Results of the node-wise aggregator study: Mean absolute error (MAE) with its 95% confidence interval.

Method		w/o feat.	hard feat.	easy feat.
MPNN	add	1.97±0.00	1.83±0.01	1.84±0.01
	mean	1.97±0.00	1.88±0.01	1.79±0.00
	max	1.96±0.00	1.89±0.00	1.85±0.00
	min	1.97±0.00	1.90±0.00	1.91±0.01
GCN		1.81±0.00	1.81±0.01	1.81±0.01
GPR-GNN		1.81±0.00	1.81±0.01	1.81±0.01
NW-GNN		**1.71±0.01**	**1.55±0.06**	**1.35±0.08**

4.2.3 Effect of Node-wise Resolution.

We construct a synthetic dataset and define a node classification task on it. At first, we generate 150 graphs, each of which is a tree, consisting of a root node, five child nodes, and twenty-five leaf nodes (five for each child). Then we randomly label each root node as either positive or negative and generate the attributes for the nodes according to its label. In detail, if the root node is a positive example, we allow fifteen leaf nodes to have attributes drawn from $\mathcal{N}(\cdot|1, 0.5)$ while the other ten have attributes drawn from $\mathcal{N}(\cdot|0, 0.5)$. Otherwise, we allow only twelves leaf nodes to have attributes drawn from $\mathcal{N}(\cdot|1, 0.5)$. If we allocate the attributes sampled from $\mathcal{N}(\cdot|1, 0.5)$ for the five child nodes as equally as possible, the graph belongs to Group1. If we allocate the attributes sampled from $\mathcal{N}(\cdot|1, 0.5)$ to the five child nodes with splits as skewed as possible, the graph belongs to Group2. Finally, we randomly split the generated graphs into equal-sized train/valid/test sets.



(a) Comparison between different resolution configurations. **(b) The predictions of resolution controller.**

Figure 5: Results of the node-wise resolution study.

To study the impact of the resolution controller, we adopt a two-layer GraphSAGE and neighbor sampler [8] as our testbed, where the fixed resolutions {1-5, 2-3, 3-2, 5-1} are considered as baselines and the search space of our resolution controller. Meanwhile, we consider both the default mean pooling and the max pooling for the GraphSAGE models for the generality reason. We repeat the training course of each method ten times, each of which consists of 20 epochs.

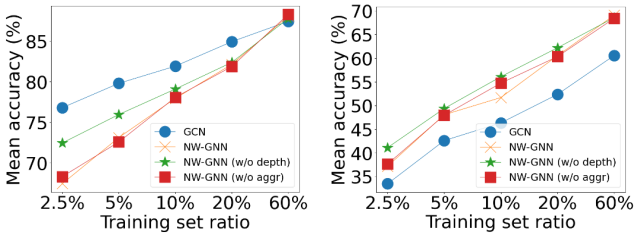
The results are shown in Fig. 5a, where NW-GNN outperforms the fixed resolutions, whichever aggregator the backbone model uses. We conduct a t-test for the results, where the improvement of NW-GNN is significant (p-value < 0.05) under the "mean pooling" setting but not significant under the "max pooling" setting.

Furthermore, we conduct a case study to understand the behavior of the resolution controller. Specifically, we collect the predictions made by our resolution controller for each graph in the valid set and then average the predictions for Group1 and Group2, respectively. We present the averaged predictions for each group at the first five epochs in Fig. 5b, where the height of each bar reflects the probability of the corresponding resolution. The bar with or without shadow corresponds to Group1 or Group2, respectively. Under both the "mean pooling" and the "max pooling" settings, the controller learns to use resolution "1-5" for Group1 while use resolution "5-1" for Group2. For the graphs in Group2, since the attributes are allocated for the child nodes unequally, the mean of the attributes in each child's leaf nodes varies a lot among the child nodes. Thus, applying resolution "5-1" for Group2 reduces the variance, which explains the improvements brought in by the resolution controller.

4.3 Further Analysis

We have validated the benefits brought in by the designed controllers. Meanwhile, as they also introduce additional trainable parameters, in this section, we study their influences from both sample efficiency and running time. We use "w/o depth" and "w/o aggr" to indicate NW-GNN with only aggregator controller or depth controller, respectively.

Sample Efficiency. The methods considered in this comparison include GCN, NW-GNN, and NW-GNN with ablation of either the depth or the aggregator controller. In addition to the "dense split" (i.e., 60%/20%/20%) used in Sec. 4.1, we also consider splits "$x/x/(1-2x)$" with $x \in \{2.5\%, 5\%, 10\%, 20\%\}$. The results are shown in Fig. 6. Overall, the performances of each method grow with the increase of training set ratio. On Cora, a typical homophilic graph, GCN benefits from its low-pass filter nature and outperforms NW-GNN when the training set is very limited. However, NW-GNN



(a) On Cora. **(b) On Chameleon.**

Figure 6: Results of the sample efficiency study.

exhibits a much steeper growth curve and overtakes GCN with the "dense split". On Chameleon, NW-GNN surpasses GCN under different settings consistently, and the slopes of their performance curves are comparable. These observations suggest that the sample efficiency of NW-GNN is at least as good as GCN, where the additional parameters introduced by our controllers will not be a burden. For the ablation study, NW-GNN leads on both the datasets under the "dense split" setting, but, with a smaller training set, the removal of a controller (e.g., the depth controller or the aggregator controller) may improve the performances. This phenomenon is caused by the decrease in the number of trainable parameters.

Running Time. We present the running time for each method in Table 5, where all training processes are executed on a NVIDIA GeForce RTX 2080 Ti GPU (12GB memory). NW-GNN is slower than other methods due to the existence of the controllers, which make inferences about the architectures before the execution of graph convolution operation. Compared to the SOTA method GPR-GNN, such increase is around 2~5 times, which is tolerable in practice, considering the performance improvement shown in Table 1.

5 CONCLUSIONS AND FUTURE DIRECTIONS

Motivated by observations with GNN applications, we propose a framework to enable the node-wise architecture for GNN models, in which the designed context-aware controllers can automatically utilize the local information of each node. A series of experiments show that the proposed framework outperforms state-of-the-art methods on six real-world datasets. Since no existing graph convolution-based model can simultaneously win on such a collection of various graphs, we can confirm that node-wise architecture can make the inflexible GNN models versatile. In this paper, We have focused on demonstrating the effects of node-wise depth, aggregator, and resolution and how the corresponding controllers work. In the future, more additional controllers can be added as other instantiations of the proposed framework, which deserves further studies.

6 ACKNOWLEDGMENTS

We thank the reviewers for their valuable comments. Zhewei Wei's work was partially done at Gaoling School of Artificial Intelligence, Peng Cheng Laboratory, Beijing Key Laboratory of Big Data Management and Analysis Methods and MOE Key Lab of Data Engineering and Knowledge Engineering. Zhewei Wei was supported in part by National Natural Science Foundation of China (No. 61932001, No. 61972401), by the major key project of PCL (PCL2021A12), Beijing Natural Science Foundation (No. 4222028) and by Alibaba Group through Alibaba Innovative Research Program.

Table 5: Average running time per epoch (ms) and (/) average total running time (s).

	Cora	CiteSeer	PubMed	Computers	Photo	Chameleon	Actor	Squirrel	Texas	Cornell
MLP	1.87/0.39	1.89/0.38	2.13/0.47	2.01/0.47	1.82/0.39	1.81/0.37	1.85/0.37	1.95/0.39	1.98/0.41	1.73/0.36
GCN	4.12/0.90	4.00/0.89	4.05/0.87	5.12/1.23	4.22/1.09	4.33/1.03	4.07/0.83	4.77/1.13	3.71/0.75	3.78/0.77
ChebyNet	5.67/1.17	7.45/1.59	7.70/1.65	26.18/5.98	13.61/3.00	11.35/2.29	7.32/1.48	44.58/9.04	4.48/0.94	4.39/0.90
GraphSAGE	3.08/0.63	4.10/0.85	5.18/1.07	18.78/4.16	8.97/1.94	7.16/1.63	4.66/0.94	30.27/6.11	2.99/0.62	2.97/0.61
GIN	3.11/0.63	4.45/0.90	5.05/1.07	17.16/5.14	9.25/3.09	7.42/1.66	4.68/1.07	30.12/11.81	3.12/0.67	3.14/0.65
GAT	5.77/1.24	5.90/1.27	6.26/1.30	6.90/1.78	5.90/1.42	5.77/1.33	5.94/1.20	6.12/1.24	5.79/1.19	5.74/1.19
JKNet	13.59/3.02	10.56/2.13	11.80/2.94	10.81/3.11	10.42/3.14	12.23/3.10	12.68/2.58	14.16/2.87	10.51/2.19	11.95/2.42
APPNP	5.52/1.19	5.97/1.31	5.74/1.23	6.34/1.57	5.80/1.55	5.84/1.19	5.75/1.16	5.98/1.21	5.69/1.17	5.70/1.18
GPR-GNN	6.42/1.31	6.63/1.35	6.59/1.33	7.31/1.73	6.62/1.56	6.5/1.52	6.82/1.49	6.68/3.86	6.69/1.46	6.86/1.42
NW-GNN w/o depth	12.26/2.68	13.25/2.68	16.44/3.70	38.62/21.43	23.06/5.55	12.10/3.08	12.02/2.43	32.46/11.60	12.25/2.48	16.41/3.42
NW-GNN w/o aggr	12.04/2.47	16.74/3.72	20.50/4.58	38.04/12.78	22.84/5.17	12.65/3.67	12.38/2.51	32.88/7.69	17.95/4.02	15.93/3.55
NW-GNN w/ both	14.07/3.29	13.75/3.22	21.34/4.76	41.44/8.96	23.37/5.39	13.89/3.47	18.48/3.76	33.87/14.27	20.48/4.81	23.11/4.90

REFERENCES

[1] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*.

[2] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *International Conference on Learning Representations*.

[3] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. 2020. Principal Neighbourhood Aggregation for Graph Nets. In *Advances in Neural Information Processing Systems*.

[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems* (2016).

[5] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2020. Graph Neural Architecture Search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*.

[6] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*.

[7] Chaoyu Guan, Xin Wang, and Wenwu Zhu. 2021. AutoAttend: Automated Attention Representation Search. In *Proceedings of the 38th International Conference on Machine Learning*.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*.

[9] Mingguo He, Zhewei Wei, Zengfeng Huang, and Hongteng Xu. 2021. BernNet: Learning Arbitrary Graph Spectral Filters via Bernstein Approximation. *arXiv preprint arXiv:2106.10994* (2021).

[10] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* (1997).

[11] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *Neural Information Processing Systems (NeurIPS)* (2020).

[12] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[13] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations (ICLR)*.

[14] Kwei-Herng Lai, Daochen Zha, Kaixiong Zhou, and Xia Hu. 2020. *Policy-GNN: Aggregation Optimization for Graph Neural Networks*.

[15] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. 2010. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World wide web*.

[16] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*.

[17] Liam Li, Mikhail Khodak, Nina Balcan, and Ameet Talwalkar. 2021. Geometry-Aware Gradient Algorithms for Neural Architecture Search. In *International Conference on Learning Representations*.

[18] Yanxi Li, Zean Wen, Yunhe Wang, and Chang Xu. 2021. One-shot Graph Neural Architecture Search with Dynamic Search Space. *Proceedings of the AAAI Conference on Artificial Intelligence* (2021).

[19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*.

[20] Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, Le Song, and Yuan Qi. 2019. Geniepath: Graph neural networks with adaptive receptive paths. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[21] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. 2022. Is Homophily a Necessity for Graph Neural Networks?. In *International Conference on Learning Representations*.

[22] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*.

[23] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*.

[24] Susheel Suresh, Vinith Budde, Jennifer Neville, Pan Li, and Jianzhu Ma. 2021. Breaking the Limit of Graph Neural Networks by Improving the Assortativity of Graphs with Local Mixing Patterns. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (2021).

[25] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*.

[26] Hao Tang, Zhiao Huang, Jiayuan Gu, Baoliang Lu, and Hao Su. 2020. Towards Scale-Invariant Graph-related Problem Solving by Iterative Homogeneous GNNs. *the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)* (2020).

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[28] Zhili Wang, Shimin Di, and Lei Chen. 2021. AutoGEL: An Automated Graph Neural Network with Explicit Link Information. *Advances in Neural Information Processing Systems* 34 (2021).

[29] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* (2021), 4–24.

[30] Yiqing Xie, Sha Li, Carl Yang, Raymond Chi-Wing Wong, and Jiawei Han. 2020. When Do GNNs Work: Understanding and Improving Neighborhood Aggregation. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*.

[31] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[32] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*.

[33] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. *GNNExplainer: Generating Explanations for Graph Neural Networks*.

[34] Jiaxuan You, Zhitao Ying, and Jure Leskovec. 2020. Design space for graph neural networks. *Advances in Neural Information Processing Systems* (2020).

[35] Ziwei Zhang, Xin Wang, and Wenwu Zhu. 2021. Automated Machine Learning on Graphs: A Survey. *arXiv preprint arXiv:2103.00742* (2021).

[36] Huan Zhao, Quanming Yao, and Weiwei Tu. 2021. Search to aggregate neighborhood for graph neural network. In *ICDE*.

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

[38] Marinka Zitnik and Jure Leskovec. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* (2017).

A MORE ON OPTIMIZATION

As we have discussed in Sec. 3.2, the forward propagation of our model involves the backbone GNN model, the depth controller, and the aggregator controller. Thus, denoting the loss function by $\mathcal{L}$, we can calculate its gradients $\nabla_{\boldsymbol{\theta}}\mathcal{L}$, $\nabla_{\boldsymbol{\phi_d}}\mathcal{L}$, and $\nabla_{\boldsymbol{\phi_a}}\mathcal{L}$, respectively. Meanwhile, for each node $v \in \mathcal{V}$, the forward propagation results in $\mathbf{z}_v$ with entire graph and results in $\hat{\mathbf{z}}_v$ with a sampled graph which involves the resolution controller. Specifically, our resolution controller rolls out a resolution configuration for each node $v \in \mathcal{V}$, where the resolution determines how to sample a neighborhood for calculating $\hat{\mathbf{z}}_v$. Thus, we can calculate reward signals to update $\boldsymbol{\phi}$ by policy gradients.

Since the parameters of our controllers play a similar role as the architecture parameter defined in differentiable NAS, we follow DARTS [19] to alternatively update $\boldsymbol{\theta}$ on the training set $\mathcal{D}_{\text{train}}$ and $\boldsymbol{\phi}$ on the valid set $\mathcal{D}_{\text{val}}$. We summarize our learning procedure in Algorithm 1.

B IMPLEMENTATION DETAILS

B.1 On Real-world Datasets

We consistently preprocess the ten datasets. First, we follow GPR-GNN [2] to normalize the attributes for each node. The splits of nodes into train/valid/test sets are conducted with different random seeds for the ten trials, where we ensure the ten seeds used for different methods are the same to keep the comparison fair.

Since the baselines are implemented with the code of GPR-GNN, we inherit their configurations mostly. For all the baselines except for NAS*, we conduct HPO for them with learning rate $\in \{0.002, 0.01, 0.05\}$, weight decay $\in \{0, 5\times10^{-4}\}$, and depth $L \in \{2, 3\}$. For NAS*, in addition to these search spaces, we allow it to select candidate aggregators for its graph convolutional layers from {min, max, add, mean, self_msg}. For NW-GNN, we use the second design of depth controller (see Sec. 3.1). Both the depth and aggregator controllers are parameterized as a MLP and fed with the node embedding calculated at the corresponding layer. In addition to learning rate and weight decay, we search for some dedicated hyper-parameters for NW-GNN, including selecting the temperature of controllers' softmax operation from $\{1.0, 10.0\}$, whether to include "self_msg" in the search space of aggregator controller, and whether to optimize the backbone GNN model and the controllers equally. The optimal hyper-parameter configurations are provided in our source code.

Each training course has 1,000 epochs, where the model is evaluated on the valid set after each epoch. The checkpoint at the epoch with the best performance on the valid set is reserved. For HPO, each configuration is evaluated by the mean (valid) accuracy without leaking the test set.

B.2 On Synthetic Datasets

Node-wise depth. First, we generate a random graph whose degree distribution follows a power law. Its basic statistics are as follow: Number of nodes is 2,467, number of edges is 19,688, average degree is 7.98, and clustering coefficient equals 0.126. Since the values of the labels span a wide range with the maximal value of 2,467 and the minimal value of 5, we take a logarithm for them for the numeric

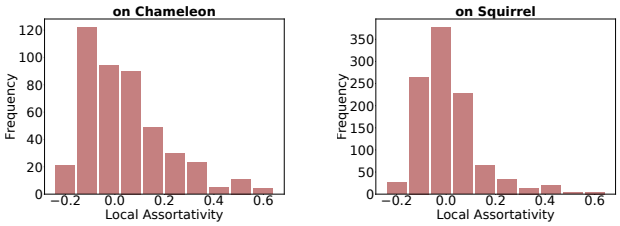


Figure 7: Distributions of the test nodes' local assortativity.

stability reason. Under the "w/o feat." setting, we use the constant "one" as the node attribute.

Under the "hard feat." setting, for each node v, we first sample a 16-dimensional attribute from $\mathcal{N}(\cdot|\mathbf{0}, 0.25\mathbf{I})$ and execute message passing for k_v iterations, where summation is adopted as the aggregator. In this way, a 16-dimensional one-hot vector can be constructed to indicate the index of maximal value in the 16-dimensional aggregated message. We concatenate the two vectors and regard it as the attribute for this node. As for "easy feat." setting, we simply assign a 4-dimensional one-hot vector for each node as its attribute, implying the ground-truth depth of this node.

Finally, we randomly split the node set of this graph into train, valid, and test set with a ratio of 40%/30%/30%.

Node-wise aggregator. The graph is randomly generated in the same way as before. Its basic statistics are as follow: Number of nodes is 2,117, number of edges is 25,298, average degree is 11.94, and clustering coefficient equals 0.094. The nodes on this graph are also splitted in the same way as before.

Table 6: Datasets properties and statistics.

	Cora	CiteSeer	PubMed	Computers	Photo	Chameleon	Actor	Squirrel	Texas	Cornell
#Nodes	2,708	3,327	19,717	13,752	7,650	2,277	5,201	7,600	183	183
#Edges	5,278	4,552	44,324	245,861	119,081	31,371	198,353	26,659	279	277
#Features	1,433	3,703	500	767	745	2,325	2,089	932	1,703	1,703
#Classes	7	6	5	10	8	5	5	5	5	5
$\mathcal{H}(G)$	0.825	0.718	0.792	0.802	0.849	0.247	0.217	0.215	0.057	0.301