PaSca: a Graph Neural Architecture Search System under the Scalable Paradigm

Wentao Zhang[1,2], Yu Shen[1], Zheyu Lin[1], Yang Li[1], Xiaosen Li[2], Wen Ouyang[2],
Yangyu Tao[2], Zhi Yang[1], Bin Cui[1,3]

[1]School of CS & Key Lab of High Confidence Software Technologies, Peking University
[2]Tencent Inc. [3]Institute of Computational Social Science, Peking University (Qingdao), China
[1]{wentao.zhang, shenyu, linzheyu, liyang.cs, yangzhi, bin.cui}@pku.edu.cn
[2]{wtaozhang, hansenli, gdpouyang, brucetao}@tencent.com

ABSTRACT

Graph neural networks (GNNs) have achieved state-of-the-art performance in various graph-based tasks. However, as mainstream GNNs are designed based on the neural message passing mechanism, they do not scale well to data size and message passing steps. Although there has been an emerging interest in the design of scalable GNNs, current researches focus on specific GNN design, rather than the general *design space*, limiting the discovery of potential scalable GNN models. This paper proposes PaSca, a new paradigm and system that offers a principled approach to systemically construct and explore the design space for scalable GNNs, rather than studying individual designs. Through deconstructing the message passing mechanism, PaSca presents a novel Scalable Graph Neural Architecture Paradigm (SGAP), together with a general architecture design space consisting of 150k different designs. Following the paradigm, we implement an auto-search engine that can automatically search well-performing and scalable GNN architectures to balance the trade-off between multiple criteria (e.g., accuracy and efficiency) via multi-objective optimization. Empirical studies on ten benchmark datasets demonstrate that the representative instances (i.e., PaSca-V1, V2, and V3) discovered by our system achieve consistent performance among competitive baselines. Concretely, PaSca-V3 outperforms the state-of-the-art GNN method JK-Net by 0.4% in terms of predictive accuracy on our large industry dataset while achieving up to 28.3× training speedups.

CCS CONCEPTS

• **Mathematics of computing → Graph algorithms**.

KEYWORDS

Graph Neural Networks, Scalable Graph Learning, Design Space

1 INTRODUCTION

Graph neural networks (GNNs) [56] have become the state-of-the-art methods in many graph representation learning scenarios such as node classification [8, 31, 32, 61], link prediction [3, 14, 47, 54], recommendation [17, 34, 53, 58], and knowledge graphs [1, 48, 49, 51]. Most GNN pipelines can be described in terms of the neural message passing (NMP) framework [15], which is based on the core idea of recursive neighborhood aggregation and transformation. Specifically, during each iteration, the representation of each node is updated (with neural networks) based on messages received from its neighbors. Since they typically need to perform a recursive neighborhood expansion to gather neural messages repeatedly, this process leads to an expensive neighborhood expansion, which grows exponentially with layers. The exponential growth of neighborhood size corresponds to an exponential IO overhead, which is the major challenge of large-scale GNN computation.

To scale up GNNs to web-scale graphs, recent work focuses on designing training frameworks with sampling approaches (e.g., DistDGL [66], NextDoor [19], SeaStar [55], FlexGraph [46], Dorylus [43], GNNAdvisor [50], etc.). Although distributed training is applied in these frameworks, they still suffer from high communication costs due to the recursive neighborhood aggregation during the training process. To demonstrate this issue, we utilize distributed training functions provided by DGL [2] to execute the train pipeline of GraphSAGE [15]. We partition the Reddit dataset across multiple machines and treat each GPU as a worker, and then calculate the speedup relative to the runtime of two workers. Figure 1 illustrates the training speedup along with the number of workers and the bottleneck in distributed settings. In particular, Figure 1(a) shows that the scalability of GraphSAGE is limited even when the mini-batch training and graph sampling method are adopted. Figure 1(b) further shows that the scalability is mainly bottlenecked by the aggregation procedure in which high data loading cost is incorporated to gather neighborhood information.

Different from the recently developed GNN systems [43, 50], we address the scalability challenges from an orthogonal perspective: re-designing the GNN pipeline to make the computing naturally scalable. To ensure scalability, we consider a different GNN training pipeline from most existing work: treating data aggregation over the graph as pre/post-processing stages that are separate from training. While there has been an emerging interest in specific architectural designs with decoupled pipeline [12, 52, 63, 68], current researches
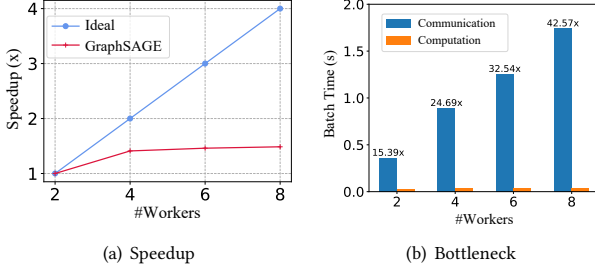
(a) Speedup (b) Bottleneck

Figure 1: The speedup and bottleneck of a two-layer Graph-SAGE along with the increased workers on Reddit dataset.

only focus on *specific GNN instances*, rather than the *general design space*, which limits the discovery of potential scalable GNN variants. In addition, new architecture search systems are required to perform extensive exploration over the design space for scalable GNNs, which is also a major motivation to our work.

To the best of our knowledge, we propose the first paradigm and system – PaSca to explore the designs of scalable GNN, which makes the following contributions:

Scalable Paradigm. We introduce the Scalable Graph Neural Architecture Paradigm (SGAP) with three operation abstractions: (1) `graph_aggregator` captures the structural information via graph aggregation operations, (2) `message_aggregator` combines different levels of structural information, and (3) `message_updater` generates the prediction based on the multi-scale features. Compared with the recently published scalable GNN systems, the SGAP interfaces in PaSca are motivated and implemented differently: (1) The APIs of GNN systems are used to express existing GNNs, whereas we propose a novel GNN pipeline abstraction to define the general design space for scalable GNN architectures. (2) The existing system contains two stages — sampling and training, where sampling is not a decoupled pre-processing stage and needs to be performed for each training iteration. By contrast, the SGAP paradigm considers propagation as pre/post-processing and does not require the expensive neighborhood expansion during training.

Design Space. Based on the proposed SGAP paradigm, we further propose a general design space consisting of 6 design dimensions, resulting in 150k possible designs of scalable GNN. We find that recently emerging scalable GNN models, such as SGC [52], SIGN [12], S^2GC [68] and GBP [6] are special instances in our design space. Instead of simply generalizing existing specific GNN designs, we propose a design space with adaptive aggregation and a complementary post-processing stage beyond what is typically considered in the literature. The extension is motivated by the observation that previous GNNs (e.g., GCN [20] and SGC) suffer from model scalability issue, as shown in Figure 2. Here we use *model scalability* to describe its capability to cope with the large-scale neighborhood with increased aggregation step k. We find the underlying reason is that their aggregation processes are restricted to a fixed-hop neighborhood and are insensitive to the actual demands of different nodes, which may lead to two limitations preventing them from unleashing their full potential: (1) long-range dependencies cannot be fully leveraged due to limited hops/layers, and (2) local information are lost due to the introduction of irrelevant nodes and unnecessary messages when the number of hops increases (i.e., over-smoothing issue [23, 33, 62]). Through extending design space with adaptive
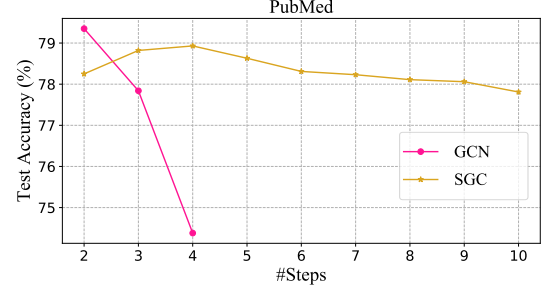

Figure 2: Test accuracy of different models along with the increased aggregation steps on PubMed dataset.

aggregators, we could discover new GNNs to achieve both model scalability and training scalability.

Auto-search Engines. We design and implement a search system that automates the search procedure for well-performing scalable GNN architectures to explore the proposed design space instead of the manual design. Our search system contains the following two engines: (1) *Suggestion engine* that implements a multi-objective search algorithm, which aims to find Pareto-optimal GNN instances given multiple criteria (e.g., predictive performance, inference time, resource consumption), allowing for a designer to select the best Pareto-optimal solution based on specific requirements; (2) *Evaluation engine* that evaluates the GNN instances from the search engine in a distributed manner. Due to the repetitive expansion in the training stage of GNNs, it is hard for existing GNN systems to scale to increasing workers. Based on the SGAP paradigm, the evaluation engine in PaSca involves the expensive neighborhood expansion only once in the pre/post-processing stages, and thus ensuring the scalability upon the number of training workers. To support the new pipeline, we implement two components: (1) the distributed graph data aggregator to pre/post-process data over graph structure, and (2) the distributed trainer where workers only need to exchange neural parameters.

Based on our auto-search system PaSca, we discover new scalable GNN instances from the proposed design space for different accuracy-efficiency requirements. Extensive experiments on ten graph datasets demonstrate the superior training scalability/efficiency and performance of searched representatives given by PaSca among competitive baselines. Concretely, the representatives (i.e., PaSca-V2 and PaSca-V3) outperform the state-of-the-art JK-Net by 0.2% and 0.4% in predictive accuracy on our industry dataset, while achieving up to 56.6× and 28.3× training speedups, respectively.

Relevance to Web. GNNs have recently been applied to a broad spectrum of web research such as social influence prediction [37, 38], network role discovery [10, 39], recommendation system [17, 54, 58], and fraud/spam detection [22, 30]. However, scalability is a major challenge that precludes GNN-based methods in practical web-scale graphs. Moreover, manually designing the well-behaved GNNs for web tasks requires immense human expertise. To bridge this gap, we highlight the relevance of the proposed PaSca platform to GNN-based web research. First, PaSca provides easier support for experts in solving their web problems via scalable GNN paradigm. Domain experts only need to provide properly formatted datasets, and PaSca can automatically search suitable and scalable GNN designs to the web-scale graphs. So PaSca permits a transition from particular GNN instances to GNN design space,

which offers exciting opportunities for scalable GNN architecture innovation. Second, PaScA provides the dis-aggregated execution pipeline for efficiently training and evaluating searched GNN models, without resorting to any approximation technique (e.g., graph sampling). Given these advancements in scalability and automaticity, our PaScA system enables practical and scalable GNN-based implementation for web-scale tasks, and thus significantly reducing the barrier when applying GNN models in web research.

2 PRELIMINARY

GNN Pipelines. Considering a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V}$, edges $\mathcal{E}$ and features for all nodes $\mathbf{x}_v \in \mathbb{R}^d, \forall v \in \mathcal{V}$, Most GNNs can be formulated using the neural message passing framework, like GCN [20], GraphSAGE [15], GAT [44], and GraphSAINT [60], where each layer adopts one aggregation and an update function. At time step t, a message vector $\mathbf{m}_v^t$ is computed with the representations of its neighbors $\mathcal{N}_v$ using an aggregate function, and $\mathbf{m}_v^t$ is then updated by a neural-network based update function:

$$\mathbf{m}_v^t \leftarrow \text{aggregate}\left(\{\mathbf{h}_u^{t-1}|u \in \mathcal{N}_v\}\right), \ \mathbf{h}_v^t \leftarrow \text{update}(\mathbf{m}_v^t). \quad (1)$$

In this way, messages are passed for K time steps in a K-layer GNN so that the steps of message passing correspond to the GNN depth. Taking the vanilla GCN [20] as an example, we have:

$$\text{GCN-aggregate}\left(\{\mathbf{h}_u^{t-1}|u \in \mathcal{N}_v\}\right) = \sum_{u \in \mathcal{N}_v} \mathbf{h}_u^{t-1}/\sqrt{\tilde{d}_v \tilde{d}_u},$$

$$\text{GCN-update}(\mathbf{m}_v^t) = \sigma(W\mathbf{m}_v^t),$$

where $\tilde{d}_v$ is the degree of node v obtained from the adjacency matrix with self-connections $\tilde{A} = I + A$. Recently, some GNN variants adopt the decoupled neural message passing (DNMP) for better graph learning. More details can be found in Appendix A.2.

Scalable GNN Instances. Following SGC [52], a recent direction for scalable GNN is to remove the non-linearity between each layer in the forward aggregation, and models in this direction have achieved state-of-the-art performance in leaderboards of Open Graph Benchmark [18]. Concretely, SIGN [12] proposes to concatenate different iterations of aggregated feature messages, while S^2GC [68] proposes a simple spectral graph convolution to average them. In addition, GBP [6] applies constants to weight aggregated feature messages of different layers. As current researches focus on studying specific architectural designs, we systematically study the architectural design space for scalable GNNs.

Graph Neural Architecture Search. As a popular direction of AutoML [16, 24, 26], neural architecture search [11, 41, 65] has been proposed to solve the labor-intensive problem of neural architecture design. Auto-GNN [67] and GraphNAS [13] are early approaches that apply reinforcement learning with RNN controllers on a fixed search space. You et al. [59] define a similarity metric and search for the best transferable architectures across tasks via random search. Based on DARTS [29], GNAS [4] proposes the differentiable searching strategy to search for GNNs with optimal message-passing step. DSS [27] also adopts the differentiable strategy but optimizes over a dynamically updated search space. PaScA differs from these works in two aspects: (1) To pursue efficiency and scalability on large graphs, PaScA searches for scalable architectures under the novel

---

Algorithm 1 Scalable graph neural architecture paradigm.

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, maximum aggregation steps for pre-processing and post-processing K_{pre}, K_{post}, feature $\mathbf{x}_v$.

Output: Prediction message $\mathbf{m}_v^{K_{post}}, \forall v \in \mathcal{V}$.

1: Initialize message set $\mathcal{M}_v = \{\mathbf{x}_v\}, \forall v \in \mathcal{V}$;
2: // **Stage 1: Pre-processing**
3: Initialize feature message $\mathbf{m}_v^0 = \mathbf{x}_v, \forall v \in \mathcal{V}$;
4: **for** $1 \le t \le K_{pre}$ **do**
5:     **for** $v \in \mathcal{V}$ **do**
6:         $\mathbf{m}_v^t \leftarrow \text{graph\_aggregator}(\mathbf{m}_{\mathcal{N}_v}^{t-1})$;
7:         $\mathcal{M}_v = \mathcal{M}_v \cup \{\mathbf{m}_v^t\}$;
8:     **end for**
9: **end for**
10: // **Stage 2: Model-training**
11: **for** $v \in \mathcal{V}$ **do**
12:     $\mathbf{c}_v \leftarrow \text{message\_aggregator}(\mathcal{M}_v)$;
13:     $\mathbf{h}_v \leftarrow \text{message\_updater}(\mathbf{c}_v)$;
14: **end for**
15: // **Stage 3: Post-processing**
16: Initialize feature message $\mathbf{m}_v^0 = \mathbf{h}_v, \forall v \in \mathcal{V}$;
17: **for** $1 \le t \le K_{post}$ **do**
18:     **for** $v \in \mathcal{V}$ **do**
19:         $\mathbf{m}_v^t \leftarrow \text{graph\_aggregator}(\mathbf{m}_{\mathcal{N}_v}^{t-1})$;
20:     **end for**
21: **end for**
22: **return** $\mathbf{m}_v^{K_{post}}, \forall v \in \mathcal{V}$;

SGAP paradigm instead of classic architectures under the message passing framework; (2) Rather than optimizing the predictive performance alone, PaScA tackles the accuracy-efficiency trade-off through multi-objective optimization, and provides architectures to meet different needs of performance and inference time.

3 PASCA ABSTRACTION

To address data and model scalability issues mentioned in Section 1, we propose a novel abstraction under which more scalable GNNs can be derived. Then we define the general design space for scalable GNNs based on the proposed abstraction.

3.1 SGAP Paradigm

Our PaScA system introduces a Scalable Graph Neural Architecture Paradigm (SGAP) for designing scalable GNNs. As shown in Algorithm 1, it consists of the following three decoupled stages:

Pre-processing. For each node v, we range the step t from 1 to K_{pre}, where K_{pre} is the maximum feature aggregation step. At each step t, we use an operator, namely graph_aggregator, to aggregate the message vector collected from the neighbors $\mathcal{N}_v$:

$$\mathbf{m}_v^t \leftarrow \text{graph\_aggregator}\left(\{\mathbf{m}_u^{t-1}|u \in \mathcal{N}_v\}\right), \quad (2)$$

where $\mathbf{m}_v^0 = \mathbf{x}_v$. The messages are passed for K_{pre} steps in total during pre-processing, and $\mathbf{m}_v^t$ at step t can gather the neighborhood information from nodes that are t-hop away (lines 4-9).

Model-training. The multi-hop messages $\mathcal{M}_v = \{\mathbf{m}_v^t \mid 0 \le t \le K_{pre}\}$ are then aggregated into a single combined message vector

$\mathbf{c}_v$ for each node v (line 12) as:

$$\mathbf{c}_v \leftarrow \text{message\_aggregator}(\mathcal{M}_v). \tag{3}$$

Note that, if the message_aggregator is not applied, the combined message vector $\mathbf{c}_v$ is set to the message of the last step $\mathbf{m}_v^{K_{pre}}$.

We then use the message_updater to learn the class distribution of all nodes, i.e., the soft predictions (softmax outputs) predicted by the updater (line 13). Specifically, PaSca applies the Multi-layer Perceptron (MLP) as the updater, and we denote the depth of MLP as the transformation step K_{trans}. It learns node embedding $\mathbf{h}_v$ from the combined message vector $\mathbf{c}_v$:

$$\mathbf{h}_v \leftarrow \text{message\_updater}(\mathbf{c}_v). \tag{4}$$

Post-processing. Motivated by Label Propagation [45] which aggregates the node labels, we regard the soft predictions as new features (line 16). Then, we use the graph_aggregator again at each step to aggregate the adjacent node predictions and make the final prediction (lines 17-21) as:

$$\mathbf{m}_v^t \leftarrow \text{graph\_aggregator}\left(\left\{\mathbf{m}_u^{t-1} | u \in \mathcal{N}_v\right\}\right), \tag{5}$$

where $\mathbf{m}_v^0 = \mathbf{h}_v$ is the original node prediction.

We introduce SGAP to address both training and model scalability challenges. Specifically, it differs from the previous NMP and DNMP framework in terms of message type, message scale, and pipeline: (1) To perform the aggregate function for the next step, existing GNNs in NMP and DNMP update the hidden state $\mathbf{h}_v^t$ by applying the message vector $\mathbf{m}_v^t$ with neural networks. By contrast, SGAP allows passing node feature messages without applying graph_aggregator on the hidden states. As a result, this message passing procedure is independent of learnable model parameters and can be easily pre-computed, thus leading to high scalability and speedups. (2) Most GNNs in NMP and DNMP only utilizes the last message vector $\mathbf{m}_v^{K_{pre}}$ to compute the final hidden state $\mathbf{h}_v^{K_{pre}}$. SGAP assumes that the optimal neighborhood expansion size should be different for each node v, and thus we retain all the messages $\{\mathbf{m}_v^t | t \in [1, K_{pre}]\}$ that a node v receives over different steps (i.e., localities). The multi-scale messages are then aggregated per node into a single vector via message_aggregator, such that we could balance the preservation of information from both local and extended (multi-hop) neighborhoods for each node. (3) Besides feature aggregation, we propose a complementary post-processing stage to aggregate predictions (soft labels), which is not typically considered in the existing literature.

3.2 Design Space under SGAP

Following the SGAP paradigm, we propose a general design space for scalable GNNs, as shown in Table 1. The design space contains three integer and three categorical parameters, which are responsible for the choice of aggregators and the steps of aggregation and transformation. Each configuration sampled from the search space represents a unique scalable architecture, resulting in 150k possible designs in total. One can also include more aggregators in the current space with future state-of-the-arts. In the following, we first introduce the aggregators used in our design space, and then explore interesting GNN instances in our defined space.

3.2.1 Graph Aggregators. To capture the information of nodes that are several hops away, PaSca adopts a graph_aggregator to combine the nodes with their neighbors during each time step. Intuitively, it is unsuitable to use a fixed graph_aggregator for each task since the choice of graph aggregators depends on the graph structure and features. Thus PaSca provides three different graph aggregators to cope with different scenarios, and one could add more aggregators following the semantic of graph_aggregator.

Augmented normalized adjacency (Aug. NA) [20]. It applies the random walk normalization on the augmented adjacency matrix $\tilde{A} = I + A$, which is simple yet effective on a range of GNNs. The normalized graph_aggregator is:

$$\mathbf{m}_v^t = \sum_{u \in \mathcal{N}_v} \frac{1}{\tilde{d}_u} \mathbf{m}_u^{t-1}. \tag{6}$$

Personalized PageRank (PPR). It focuses on its local neighborhood using a restart probability $\alpha \in (0, 1]$ and performs well on graphs with noisy connectivity. While the calculation of the fully personalized PageRank matrix is computationally expensive, we apply its approximate computation [21]:

$$\mathbf{m}_v^t = \alpha \mathbf{m}_v^0 + (1 - \alpha) \sum_{u \in \mathcal{N}_v} \frac{1}{\sqrt{\tilde{d}_v \tilde{d}_u}} \mathbf{m}_u^{t-1}, \tag{7}$$

where the restart probability α allows to balance preserving locality (i.e., staying close to the root node to avoid over-smoothing) and leveraging the information from a large neighborhood.

Triangle-induced adjacency (Triangle. IA) [35]. It accounts for the higher-order structures and helps distinguish strong and weak ties on complex graphs like social graphs. We assign each edge a weight representing the number of different triangles it belongs to, which forms a weight matrix A^{tri}. We denote d_v^{tri} as the degree of node v from the weighted adjacency matrix A^{tri}. The aggregator is then calculated by applying a row-wise normalization:

$$\mathbf{m}_v^t = \sum_{u \in \mathcal{N}_v} \frac{1}{d_v^{tri}} \mathbf{m}_u^{t-1}. \tag{8}$$

3.2.2 Message Aggregators. Before updating the hidden state of each node, PaSca proposes to apply a message_aggregator to combine messages obtained by graph_aggregator per node into a single vector, such that the subsequent model learns from the multi-scale neighborhood of a given node. We summarize the different message aggregators PaSca as follows,

Non-adpative aggregator. This type of aggregator does not consider the correlation between messages and the center node. The messages are directly concatenated or summed up with weights to obtain the combined message vector as,

$$c_{msg} \leftarrow \oplus_{\mathbf{m}_v^i \in M_v} w_i f(\mathbf{m}_v^i), \tag{9}$$

where f is a function used to reduce the dimension of message vectors, and $\oplus$ can be concatenating or pooling operators including average pooling or max pooling. Note that, for aggregator type "Mean", "Max" and "Concatenate", each weight w_i is set to 1 for each message. For aggregator type "Weighted", we set the weight to constants following GBP [6]. Compared with pooling operators, though the concatenating operator keeps all the input message

Table 1: The search space for scalable GNNs in our PaSca system.

Stages	Name	Range/Choices	Type
Pre-processing	Aggregation steps (K_{pre})	[0, 10]	Integer
	Graph aggregators (GA_{pre})	{Aug.NA, PPR($\alpha = 0.1$), PPR($\alpha = 0.2$), PPR($\alpha = 0.3$), Triangle. IA}	Categorical
Model training	Message aggregators (MA)	{None, Mean, Max, Concatenate, Weighted, Adaptive}	Categorical
	Transformation steps (K_{trans})	[1, 10]	Integer
Post-processing	Aggregation steps (K_{post})	[0, 10]	Integer
	Graph aggregators (GA_{post})	{Aug.NA, PPR($\alpha = 0.1$), PPR($\alpha = 0.2$), PPR($\alpha = 0.3$), Triangle. IA}	Categorical

Figure 3: The influence of aggregation steps on 20 randomly sampled nodes on Citeseer dataset.

Table 2: Current scalable GNNs in our design space.

Models	Pre-processing	Model training		Post-processing
	GA_{pre}	MA	K_{trans}	GA_{post}
SGC	Aug.NA	None	1	/
SIGN	Optional	Concatenate	1	/
S²GC	PPR	Mean	1	/
GBP	Aug.NA	Weighted	≥ 2	/
PaSca-APPNP	/	/	≥ 2	PPR

information, the dimension of its outputs increases as K_{pre} grows, leading to additional computational cost in the downstream updater.

Adpative aggregators. The messages of different hops make different contributions to the final performance. As shown in Figure 3, we apply GCN with different layers to conduct node classification on Citeseer. Note that the X-axis is the node id and Y-axis is the aggregation steps (number of layers in GCN). The color from white to blue represents the ratio of being predicted correctly in 50 different runs. We observe that most nodes are well classified with two steps, and as a result, most carefully designed GNN models are set with two layers (i.e., steps). In addition, the predictive accuracy on 13 of the 20 sampled nodes increases with a certain step larger than two. This motivates the design of *node-adaptive* aggregation functions, which determines the importance of a node's message at different ranges rather than fixing the same weights for all nodes.

To this end, we propose the gating aggregator, which generates retainment scores that indicate how much the corresponding messages should be retained in the final combined message.

$$\mathbf{c_{msg}} \leftarrow \sum_{\mathbf{m}_v^i \in M_v} w_i \mathbf{m}_v^i, \; w_i = \sigma(\mathbf{sm}_v^i), \tag{10}$$

where $\mathbf{s}$ is a trainable vector to generate gating scores, and σ is the sigmoid function. With the adaptive `message_aggregator`, the model can balance the messages from the multi-scale neighborhoods for each node at the expense of training extra parameters.

3.3 SGAP Instances

Recent scalable GNN models, such as SGC [52], SIGN [12], S²GC [68] and GBP [6], can be considered as specific instances in our defined design space, as shown in Table 2. We see that most current scalable GNNs ignore the post-processing stages, which can boost the model performance demonstrated by our experiments.

Besides, various scalable GNNs can be obtained by using different design choices under SGAP. For example, the current state-of-the-art scalable model GBP sets the `graph_aggregator` as Aug.NA and uses the weighted strategy in the `message_aggregator`. We decouple MLP training and information propagation in APPNP [21] into two individual processes and get a new scalable model PaSca-APPNP. *To effectively explore the large design space, we implement*

an auto-search system engine below to automate the search procedure of scalable GNN architecture instead of manual design.

4 PASCA ENGINES

Figure 4 shows the overview of our proposed auto-search system to explore GNN designs under PaSca abstraction. It consists of two engines: the *search* engine and the *evaluation* engine. The search engine includes the proposed designed search space for scalable GNNs and implements a suggestion server that is responsible for suggesting architectures to evaluate. The evaluation engine receives an instance and trains the corresponding architecture in a distributed fashion. An iteration of the searching process is as follows: 1) The suggestion server samples an architecture instance based on its built-in optimization algorithm and sends it to the evaluation engine; 2) The evaluation engine evaluates the architecture and updates the suggestion server with its observed performance.

4.1 Search Engine

While prior researches on scalable GNNs [6, 12] focus on optimizing the classification error, recent applications not only require high predictive performance but also *low resource-consumption*, e.g. model size or inference time. In addition, there is typically an implicit trade-off between predictive performance and resource consumption. To this end, the suggestion server implements a *multi-objective* search algorithm to tackle this trade-off.

Concretely, we use the Bayesian optimization based on EHVI [9], a widely-used algorithm that maximizes the predicted improvement of hypervolume indicator of Pareto-optimal points relative to a given reference point. The suggestion server then optimizes over the search space following a typical Bayesian optimization as 1) Based on the observation history, the server trains multiple surrogates, namely the Gaussian Process, to model the relationships between each architecture instance and its objective values; 2) The server randomly samples a number of new instances, and suggests the best one which maximizes the EHVI based on the predicted outputs of trained surrogates; 3) The server receives the results of the suggested instance and updates its observation history.
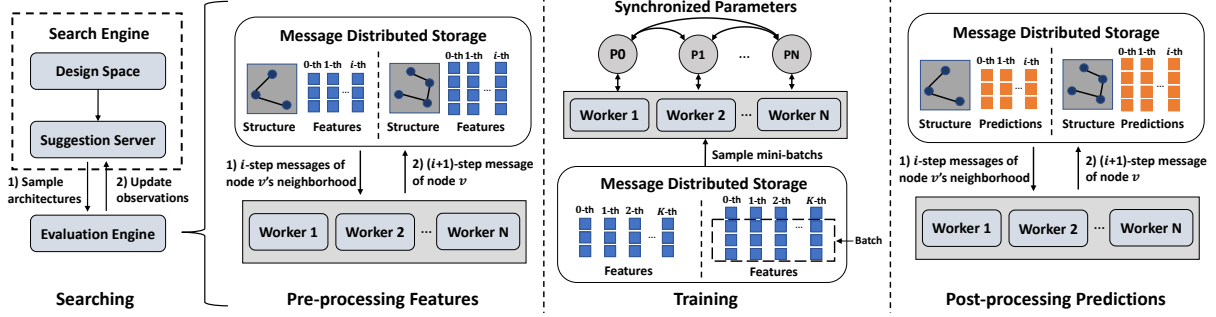
Figure 4: The workflow of PaSca, which consists of the searching and evaluation engine.

4.2 Evaluation Engine

Different from the sampling-training process of existing GNN systems (e.g., DistDGL [66], NextDoor [19], and FlexGraph [46]), the process of PaSca evaluation engine is decoupled into pre-processing, training and post-processing as illustrated in Figure 4: PaSca first *pre-computes* the feature messages for each node over the graph, and then it *combines* the messages and *trains* the model parameters with parameter sever. Finally, PaSca *post-computes* the prediction messages for each node over the graph. All the messages are partitioned and stored in a distributed storage system, and the stages can be implemented in a distributed fashion. Specifically, the engine consists of the following two components:

Graph Data Aggregator. This component handles pre-processing and post-processing stages on data aggregation over graph structure. The two stages share the same pipeline but take different messages as inputs (features for pre-processing and predictions for post-processing). We implement an efficient batch processing pipeline over distributed graph storage: The nodes are partitioned into batches, and the computation of each batch is implemented by workers in parallel with matrix multiplication. As shown in Figure 4, for each node in a batch, we firstly pull all the i-th step messages of its 1-hop neighbors from the message distributed storage and then compute the $(i + 1)$-th step messages of the batch in parallel. Next, We push these aggregated messages back for reuse in the calculation of the $(i + 2)$-th step messages. In our implementation, we treat GPUs as workers for fast processing, and the graph data are partitioned and stored on host memory across machines. Given the parallel message computation, our implementation could scale to large graphs and significantly reduce the runtime.

Neural Architecture Trainer. This component handles the training of neural networks. We optimize the parameters of each architecture with distributed SGD. The model parameters are stored on a parameter server, and multiple workers (GPUs) process the data in parallel. We adopt asynchronous training to avoid the communication overhead between workers. Each worker fetches the most up-to-date parameters and computes the gradients for a mini-batch of data, independent of the other workers.

5 EXPERIMENTS

5.1 Experimental Settings

Datasets. We conduct the experiments on three citation networks (Citeseer, Cora, and PubMed) in [20], two social networks (Flickr

Table 3: Scalable GNNs found by PaSca.

Models	Pre-processing			Model training	Post-processing	
	GA_{pre}	MA	K_{pre}	K_{trans}	GA_{post}	K_{post}
PaSca-V1	PPR(α = 0.1)	Weighted	3	2	/	/
PaSca-V2	Aug.NA	Adaptive	6	2	/	/
PaSca-V3	Aug.NA	Adaptive	6	3	PPR (α = 0.3)	4

and Reddit) in [60], four co-authorship graphs (Amazon and Coauthor) in [36], the co-purchasing network (ogbn-products) in [18] and one short-form video recommendation graph (Industry) from our industrial cooperative enterprise. Table 6 in Appendix A.1 provides the overview of the used graph datasets.

Parameters and Environment. To eliminate random factors, we run each method 20 times and report the mean and variance of the performance. More details for experimental setups and reproduction are provided in Appendix A.4 and A.5.

Baselines. In the transductive settings, we compare the searched scalable GNNs with GCN [20], GAT [44], JK-Net [57], Res-GCN [20], APPNP [21], AP-GCN [42], SGC [52], SIGN [12], S²GC [68] and GBP [6], which are SOTA models of different message passing types. In the inductive settings, the compared baselines are GraphSAGE [15], FastGCN [5], ClusterGCN [7] and GraphSAINT [60]. More descriptions about the baselines are provided in Appendix A.3.

In the following, we first analyze the superiority of representative instances searched by PaSca. Then we evaluate the transferability, training efficiency, and model scalability of PaSca representatives compared with competitive state-of-the-art baselines.

5.2 Searched Representatives

We apply the multi-objective optimization targeting at classification error and inference time on Cora. Figure 6 demonstrates the Pareto Front found by PaSca with a budget of 2000 evaluations, together with the results of several manually designed scalable GNNs. The inference time has been normalized based on instances with the minimum and maximum inference time in our design space. Interestingly, we observe that GBP and PaSca-APPNP, our extended variant of APPNP (see Table 2), falls on the Pareto Front, which indicates the superior design of the "Weighted" message_aggregator and "PPR" graph_aggregator. We also choose other three instances from the Pareto Front as PaSca-V1 to V3 with different accuracy-efficiency requirements as searched representatives of SGAP for the following evaluations. The corresponding parameters of each architecture are shown in Table 3. Among the three architectures, PaSca-V1 Pareto-dominates the other baselines except GBP, and

(a) Stand-alone on Reddit (b) Distributed on Reddit (c) Stand-alone on ogbn-product (d) Distributed on ogbn-product
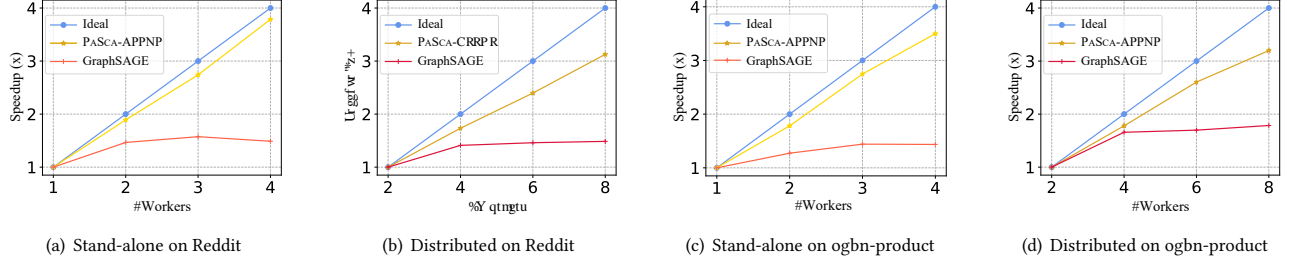
Figure 5: Scalability comparison on Reddit and ogbn-product datasets. The stand-alone scenario means the graph has only one partition stored on a multi-GPU server, whereas the distributed scenario means the graph is partitioned and stored on multi-servers. In the distributed scenario, we run two workers per machine.
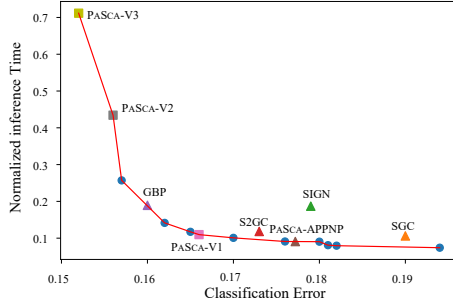

Figure 6: Pareto Front found on Cora.

Figure 7: Test accuracy over training time on Industry.

PaSca-V3 is the architecture with the best predictive performance found by the search engine.

5.3 Training Scalability

The main characteristic of our proposed design space is that the architectures sampled from the space, namely PaSca-SGAP, share high scalability upon workers. To examine the scalability of PaSca-SGAP, we choose PaSca-APPNP as a representative and compare it with GraphSAGE, a widely-used method in industry on two large-scale datasets. We train GraphSAGE with DGL and PaSca-APPNP with the evaluation engine of PaSca, respectively. We train both methods in stand-alone and distributed scenarios and then measure their corresponding speedups. The batch size is 8192 for Reddit and 16384 for ogbn-product, and the speedup is calculated by runtime per epoch relative to that of one worker in the stand-alone scenario and two workers in the distributed scenario. Without considering extra cost, the speedup will increase linearly in an ideal condition.

The corresponding results are shown in Figure 5. Since GraphSAGE requires aggregating the neighborhood nodes during training, GraphSAGE trained with DGL meets the I/O bottleneck when transmitting a large number of required neural messages. Thus, the speedup of GraphSAGE training increases slowly as the number of workers grows, which is less than 2× even with four workers in the stand-alone scenario and eight workers in the distributed scenario. Recall that the only communication cost of PaSca-SGAP is to synchronize parameters among different workers, which is essential to all distributed training methods. As a result, PaSca-SGAP trained with the evaluation engine scales up close to the ideal circumstance in both scenarios, indicating the superiority of PaSca.
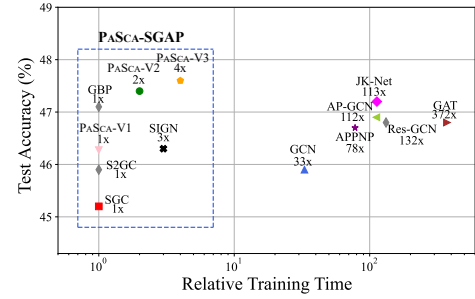
5.4 Performance-Efficiency Analysis

To test the transferability and training efficiency of PaSca models, we further evaluate them on more datasets compared with competitive baselines. The results are summarized in Table 4 and 5.

We observe that PaSca models obtain quite competitive performance in both transductive and inductive settings. In transductive settings, our simplified variant PaSca-V1 also achieves the best performance among Non-SGAP baselines on most datasets, which shows the superiority of SGAP design. In addition, PaSca-V2 and V3 outperform the best baseline GBP by a margin of 0.1%~0.6% and 0.2%~1.3% on each dataset. We attribute this improvement to the application of the adaptive `message_aggregator`. In inductive settings, Table 5 shows that PaSca-V3 outperforms the best baseline GraphSAINT by a margin of 1.1% on Flickr and 0.1% on Reddit.

We also evaluate the training efficiency of each method in the real production environment. Figure 7 illustrates the performance over training time on Industry. In particular, we pre-compute the feature messages of each scalable method, and the training time takes into account the pre-computation time. We observe that NMP architectures require at least a magnitude of training time than PaSca-SGAP. Among considered baselines, PaSca-V3 achieves the best performance with 4× training time compared with GBP and PaSca-V1. Note that, though PaSca-V1 requires the same training time as GBP, its inference time is less than GBP. Therefore, we recommend choosing PaSca-V1 to V3, along with GBP, according to different requirements of predictive performance, training efficiency, and inference time.

5.5 Model Scalability

We observe that both PaSca-V2 and V3 found by the search engine contain the "Adaptive" `message_aggregator`. In this subsection,

Table 4: Test accuracy (%) in transductive settings. "NMP" and "DNMP" refer to architectures following NMP and DNMP paradigm. "SGAP" refers to architectures following the scalable graph archtecture paradigm proposed in Section 3.1.
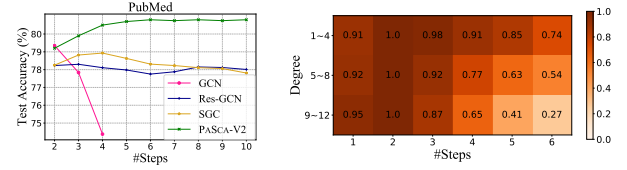
Type	Models	Cora	Citeseer	PubMed	Amazon Computer	Amazon Photo	Coauthor CS	Coauthor Physics	Industry
NMP	GCN	81.8±0.5	70.8±0.5	79.3±0.7	82.4±0.4	91.2±0.6	90.7±0.2	92.7±1.1	45.9±0.4
	GAT	83.0±0.7	72.5±0.7	79.0±0.3	80.1±0.6	90.8±1.0	87.4±0.2	90.2±1.4	46.8±0.7
	JK-Net	81.8±0.5	70.7±0.7	78.8±0.7	82.0±0.6	91.9±0.7	89.5±0.6	92.5±0.4	47.2±0.3
	ResGCN	82.2±0.6	70.8±0.7	78.3±0.6	81.1±0.7	91.3±0.9	87.9±0.6	92.2±1.5	46.8±0.5
DNMP	APPNP	83.3±0.5	71.8±0.5	80.1±0.2	81.7±0.3	91.4±0.3	92.1±0.4	92.8±0.9	46.7±0.6
	AP-GCN	83.4±0.3	71.3±0.5	79.7±0.3	83.7±0.6	92.1±0.3	91.6±0.7	93.1±0.9	46.9±0.7
SGAP	SGC	81.0±0.2	71.3±0.5	78.9±0.5	82.2±0.9	91.6±0.7	90.3±0.5	91.7±1.1	45.2±0.3
	SIGN	82.1±0.3	72.4±0.8	79.5±0.5	83.1±0.8	91.7±0.7	91.9±0.3	92.8±0.8	46.3±0.5
	S^{2}GC	82.7±0.3	73.0±0.2	79.9±0.3	83.1±0.7	91.6±0.6	91.6±0.6	93.1±0.8	45.9±0.4
	GBP	83.9±0.7	72.9±0.5	80.6±0.4	83.5±0.8	92.1±0.8	92.3±0.4	93.3±0.7	47.1±0.6
	PaSca-V1	83.4±0.5	72.2±0.5	80.5±0.4	83.7±0.7	92.1±0.7	91.9±0.3	93.2±0.6	46.3±0.4
	PaSca-V2	84.4±0.3	73.1±0.3	80.7±0.7	84.1±0.7	92.4±0.7	92.6±0.4	93.6±0.8	47.4±0.6
	PaSca-V3	**84.6±0.6**	**73.4±0.5**	**80.8±0.6**	**84.8±0.7**	**92.7±0.8**	**92.8±0.5**	**93.8±0.9**	**47.6±0.3**

Table 5: Test accuracy (%) in inductive settings.

Models	Flickr	Reddit
GraphSAGE	50.1±1.3	95.4±0.0
FastGCN	50.4±0.1	93.7±0.0
ClusterGCN	48.1±0.5	95.7±0.0
GraphSAINT	51.1±0.1	96.6±0.1
PaSca-V1	51.2±0.3	95.8±0.1
PaSca-V2	51.8±0.3	96.3±0.0
PaSca-V3	**52.1±0.2**	**96.7±0.1**

Figure 8: Left: Test accuracy of different models along with the increased aggregation steps on PubMed. Right: PaSca-V2's average gating weights of graph messages of different steps on 60 randomly selected nodes from PubMed.

we aim to explain the advantage of adaptive `message_aggregator` in the perspective of model scalability on message passing steps. We plot the changes of model performance along with the message passing steps in the left subfigure of Figure 8. For a fair comparison, we use PaSca-V2 which does not include post-processing. The vanilla GCN gets the best results with two aggregation steps, but its performance drops rapidly along with the increased steps due to the over-smoothing issue. Both Res-GCN and SGC show better performance than GCN with larger aggregation steps. Take Res-GCN as an example, it carries information from the previous step by introducing the residual connections and thus alleviates this problem. However, these two methods cannot benefit from deep GNN architecture since they are unable to balance the needs of preserving locality (i.e., staying close to the root node to avoid over-smoothing) and leveraging the information from a large neighborhood. In contrast, PaSca-V2 achieves consistent improvement and remains nondecreasing across steps, which indicates that PaSca can scales to large depth. The reason is that the adaptive `message_aggregator` in PaSca-V2 is able to adaptively and effectively combine multiscale neighborhood messages for each node.

To demonstrate this, the right subfigure of Figure 8 shows PaSca-V2's average gating weights of feature messages according to the number of steps and degrees of input nodes, where the maximum step is 6. In this experiment, we randomly select 20 nodes for each degree range (1-4, 5-8, 9-12) and plot the relative weight based

on the maximum value. We get two observations from the heat map: 1) The 1-step and 2-step graph messages are always of great importance, which shows that the adaptive `message_aggregator` captures the local information as those widely used 2-layer GNNs do; 2) The weights of graph messages with larger steps drop faster as the degree grows, which indicates that the attention-based aggregator could prevent high-degree nodes from including excessive irrelevant nodes which lead to over-smoothing. From the two observations, we conclude that the adaptive `message_aggregator` can identify the different message-passing demands of nodes and explicitly weight each graph message.

6 CONCLUSION

In this paper, we proposed PaSca, a new auto-search system that offers a principled approach to systemically construct and explore the design space for scalable GNNs, rather than studying individual designs. Experiments on ten real-world benchmarks demonstrate that the representative instances searched by PaSca outperform SOTA GNNs in terms of performance, efficiency, and scalability. PaSca can help researchers understand design choices when developing new scalable GNN models, and serve as a system to support extensive exploration over the design space for scalable GNNs.

ACKNOWLEDGMENTS

REFERENCES

[1] Anson Bastos, Abhishek Nadgeri, Kuldeep Singh, Isaiah Onando Mulang, Saeedeh Shekarpour, Johannes Hoffart, and Manohar Kaul. 2021. RECON: relation extraction using knowledge graph context in a graph neural network. In *Proceedings of the Web Conference 2021*. 1673–1685.

[2] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

[3] Lei Cai and Shuiwang Ji. 2020. A multi-scale approach for graph link prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3308–3315.

[4] Shaofei Cai, Liang Li, Jincan Deng, Beichen Zhang, Zheng-Jun Zha, Li Su, and Qingming Huang. 2021. Rethinking graph neural architecture search from message-passing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6657–6666.

[5] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *International Conference on Learning Representations*.

[6] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable graph neural networks via bidirectional propagation. *Advances in neural information processing systems* 33 (2020), 14556–14566.

[7] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 257–266.

[8] Hande Dong, Jiawei Chen, Fuli Feng, Xiangnan He, Shuxian Bi, Zhaolin Ding, and Peng Cui. 2021. On the equivalence of decoupled graph convolution network and label propagation. In *Proceedings of the Web Conference 2021*. 3651–3662.

[9] Michael Emmerich. 2005. Single-and multi-objective evolutionary design optimization assisted by gaussian random field metamodels. *University of Dormund* (2005).

[10] Changjun Fan, Li Zeng, Yuhui Ding, Muhao Chen, Yizhou Sun, and Zhong Liu. 2019. Learning to Identify High Betweenness Centrality Nodes from Scratch: A Novel Graph Neural Network Approach. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Managemenat, CIKM 2019, Beijing, Nov 3-7, 2019*. 559–568.

[11] Jiemin Fang, Yukang Chen, Xinbang Zhang, Qian Zhang, Chang Huang, Gaofeng Meng, Wenyu Liu, and Xinggang Wang. 2021. EAT-NAS: Elastic architecture transfer for accelerating large-scale neural architecture search. *Science China Information Sciences* 64, 9 (2021), 1–13.

[12] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Benjamin Chamberlain, Michael Bronstein, and Federico Monti. 2020. SIGN: Scalable Inception Graph Neural Networks. In *ICML 2020 Workshop on Graph Representation Learning and Beyond*.

[13] 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 2020*, Christian Bessiere (Ed.). ijcai.org, 1403–1409.

[14] Qipeng Guo, Xipeng Qiu, Xiangyang Xue, and Zheng Zhang. 2021. Syntax-guided text generation via graph neural network. *Sci. China Inf. Sci.* 64, 5 (2021).

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[16] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems* 212 (2021), 106622.

[17] Cheng Hsu and Cheng-Te Li. 2021. RetaGNN: Relational temporal attentive graph neural networks for holistic sequential recommendation. In *Proceedings of the Web Conference 2021*. 2968–2979.

[18] 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. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[19] Abhinav Jangda, Sandeep Polisetty, Arjun Guha, and Marco Serafini. 2021. Accelerating graph sampling for graph machine learning using GPUs. In *Proceedings of the Sixteenth European Conference on Computer Systems*. 311–326.

[20] Thomas N Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

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

[22] Ao Li, Zhou Qin, Runshi Liu, Yiqun Yang, and Dong Li. 2019. Spam Review Detection with Graph Convolutional Networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Managemenat, CIKM 2019, Beijing, Nov 3-7, 2019*. 2703–2711.

[23] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[24] Yang Li, Yu Shen, Jiawei Jiang, Jinyang Gao, Ce Zhang, and Bin Cui. 2021. MFES-HB: Efficient Hyperband with Multi-Fidelity Quality Measurements. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. AAAI Press, 8491–8500.

[25] Yang Li, Yu Shen, Wentao Zhang, Yuanwei Chen, Huaijun Jiang, Mingchao Liu, Jiawei Jiang, Jinyang Gao, Wentao Wu, Zhi Yang, et al. 2021. Openbox: A generalized black-box optimization service. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3209–3219.

[26] Yang Li, Yu Shen, Wentao Zhang, Jiawei Jiang, Bolin Ding, Yaliang Li, Jingren Zhou, Zhi Yang, Wentao Wu, Ce Zhang, et al. 2021. VolcanoML: speeding up end-to-end AutoML via scalable search space decomposition. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2167–2176.

[27] Yanxi Li, Zean Wen, Yunhe Wang, and Chang Xu. 2021. One-shot graph neural architecture search with dynamic search space. In *Proc. AAAI Conf. Artif. Intell*, Vol. 35. 8510–8517.

[28] Renjie Liao, Marc Brockschmidt, Daniel Tarlow, Alexander Gaunt, Raquel Urtasun, and Richard Zemel. 2018. Graph Partition Neural Networks for Semi-Supervised Classification. In *International Conference on Learning Representations*.

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

[30] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. 2021. Pick and choose: a GNN-based imbalanced learning approach for fraud detection. In *Proceedings of the Web Conference 2021*. 3168–3177.

[31] Xiaojun Ma, Junshan Wang, Hanyue Chen, and Guojie Song. 2021. Improving Graph Neural Networks with Structural Adaptive Receptive Fields. In *Proceedings of the Web Conference 2021*. 2438–2447.

[32] Xupeng Miao, Nezihe Merve Gürel, Wentao Zhang, Zhichao Han, Bo Li, Wei Min, Susie Xi Rao, Hansheng Ren, Yinan Shan, Yingxia Shao, et al. 2021. Degnn: Improving graph neural networks with graph decomposition. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1223–1233.

[33] Xupeng Miao, Wentao Zhang, Yingxia Shao, Bin Cui, Lei Chen, Ce Zhang, and Jiawei Jiang. 2021. Lasagne: A multi-layer graph convolutional network framework via node-aware deep architecture. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[34] Federico Monti, Michael Bronstein, and Xavier Bresson. 2017. Geometric matrix completion with recurrent multi-graph neural networks. *Advances in neural information processing systems* 30 (2017).

[35] Federico Monti, Karl Otness, and Michael M Bronstein. 2018. Motifnet: a motif-based graph convolutional network for directed graphs. In *2018 IEEE Data Science Workshop (DSW)*. IEEE, 225–228.

[36] 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*.

[37] Jinghua Piao, Guozhen Zhang, Fengli Xu, Zhilong Chen, and Yong Li. 2021. Predicting Customer Value with Social Relationships via Motif-based Graph Attention Networks. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, 3146–3157.

[38] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. 2018. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2110–2119.

[39] Benedek Rozemberczki, Peter Englert, Amol Kapoor, Martin Blais, and Bryan Perozzi. 2021. Pathfinder Discovery Networks for Neural Message Passing. In *Proceedings of the Web Conference 2021*. 2547–2558.

[40] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).

[41] Yu Shen, Yang Li, Jian Zheng, Wentao Zhang, Peng Yao, Jixiang Li, Sen Yang, Ji Liu, and Cui Bin. 2021. ProxyBO: Accelerating Neural Architecture Search via Bayesian Optimization with Zero-cost Proxies. *arXiv preprint arXiv:2110.10423* (2021).

[42] Indro Spinelli, Simone Scardapane, and Aurelio Uncini. 2020. Adaptive propagation graph convolutional network. *IEEE Transactions on Neural Networks and Learning Systems* 32, 10 (2020), 4755–4760.

[43] John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, et al. 2021. Dorylus: Affordable, Scalable, and Accurate {GNN} Training with Distributed {CPU} Servers and Serverless Threads. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. 495–514.

[44] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[45] Fei Wang and Changshui Zhang. 2007. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering* 20, 1 (2007), 55–67.

[46] Lei Wang, Qiang Yin, Chao Tian, Jianbang Yang, Rong Chen, Wenyuan Yu, Zihang Yao, and Jingren Zhou. 2021. FlexGraph: a flexible and efficient distributed framework for GNN training. In *Proceedings of the Sixteenth European Conference*

on Computer Systems. 67–82.

[47] Ruijia Wang, Shuai Mou, Xiao Wang, Wanpeng Xiao, Qi Ju, Chuan Shi, and Xing Xie. 2021. Graph structure estimation neural networks. In *Proceedings of the Web Conference 2021.* 342–353.

[48] Shen Wang, Xiaokai Wei, Cícero Nogueira dos Santos, Zhiguo Wang, Ramesh Nallapati, Andrew O. Arnold, Bing Xiang, Philip S. Yu, and Isabel F. Cruz. 2021. Mixed-Curvature Multi-Relational Graph Neural Network for Knowledge Graph Completion. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021.* 1761–1771.

[49] Xiaolong Wang, Yufei Ye, and Abhinav Gupta. 2018. Zero-shot recognition via semantic embeddings and knowledge graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 6857–6866.

[50] Yuke Wang, Boyuan Feng, Gushu Li, Shuangchen Li, Lei Deng, Yuan Xie, and Yufei Ding. 2021. GNNAdvisor: An Adaptive and Efficient Runtime System for GNN Acceleration on GPUs. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21).* 515–531.

[51] Zhouxia Wang, Tianshui Chen, Jimmy Ren, Weihao Yu, Hui Cheng, and Liang Lin. 2018. Deep reasoning with knowledge graph for social relationship understanding. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence.* 1021–1028.

[52] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning.* PMLR, 6861–6871.

[53] Shiwen Wu, Fei Sun, Wentao Zhang, and Bin Cui. 2020. Graph neural networks in recommender systems: a survey. *arXiv preprint arXiv:2011.02260* (2020).

[54] Wei Wu, Bin Li, Chuan Luo, and Wolfgang Nejdl. 2021. Hashing-accelerated graph neural networks for link prediction. In *Proceedings of the Web Conference 2021.* 2910–2920.

[55] Yidi Wu, Kaihao Ma, Zhenkun Cai, Tatiana Jin, Boyang Li, Chenguang Zheng, James Cheng, and Fan Yu. 2021. Seastar: vertex-centric programming for graph neural networks. In *Proceedings of the Sixteenth European Conference on Computer Systems.* 359–375.

[56] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* (2020).

[57] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning.* PMLR, 5453–5462.

[58] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 974–983.

[59] Jiaxuan You, Zhitao Ying, and Jure Leskovec. 2020. Design space for graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 17009–17021.

[60] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *International Conference on Learning Representations.*

[61] Wentao Zhang, Yuezihan Jiang, Yang Li, Zeang Sheng, Yu Shen, Xupeng Miao, Liang Wang, Zhi Yang, and Bin Cui. 2021. ROD: reception-aware online distillation for sparse graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining.* 2232–2242.

[62] Wentao Zhang, Zeang Sheng, Yuezihan Jiang, Yikuan Xia, Jun Gao, Zhi Yang, and Bin Cui. 2021. Evaluating deep graph neural networks. *arXiv preprint arXiv:2108.00955* (2021).

[63] Wentao Zhang, Mingyu Yang, Zeang Sheng, Yang Li, Wen Ouyang, Yangyu Tao, Zhi Yang, and Bin Cui. 2021. Node Dependent Local Smoothing for Scalable Graph Learning. *Advances in Neural Information Processing Systems* 34 (2021).

[64] Wentao Zhang, Ziqi Yin, Zeang Sheng, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. 2021. Graph attention multi-layer perceptron. *arXiv preprint arXiv:2108.10097* (2021).

[65] Ziwei Zhang, Xin Wang, and Wenwu Zhu. 2021. Automated Machine Learning on Graphs: A Survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence.* 4704–4712.

[66] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. 2020. Distdgl: distributed graph neural network training for billion-scale graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3).* IEEE, 36–44.

[67] 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).

[68] Hao Zhu and Piotr Koniusz. 2021. Simple spectral graph convolution. In *International Conference on Learning Representations.*

A APPENDIX

A.1 Dataset description

Cora, **Citeseer**, and **Pubmed**[1] are three well-known citation network datasets, and we follow the same training/validation/test split as GCN [20].

Reddit is a social network modeling the community structure of Reddit posts. This dataset is often used for inductive training, and the training/validation/test split is coherent with that of Graph-SAGE [15].

Flickr originates from NUS-wide [2] and contains different types of images based on the descriptions and common properties of online images. We use a public version of Reddit and Flickr provided by GraphSAINT[3].

Amazon Computers and **Amazon Photo** are segments of the Amazon co-purchase graph [40], where nodes represent goods, edges indicate that two goods are frequently bought together, node features are bag-of-words encoded product reviews, and class labels are given by the product category.

Coauthor CS and **Coauthor Physics** are co-authorship graph based on the Microsoft Academic Graph from the KDD Cup 2016 challenge[4]. Here, nodes are authors, that are connected by an edge if they co-authored a paper; node features represent paper keywords for each author's papers, and class labels indicate the most active fields of study for each author. We use a pre-divided version of these datasets through the Deep Graph Library (DGL)[5].

ogbn-products is an unweighted graph representing an Amazon product co-purchase network. Each node represents a product sold on Amazon, and edges between two products indicate that the products are purchased together. We use the public data split for this dataset as in Open Graph Benchmark[6].

Industry is a user-video graph collected from a real-world mobile application from our industry partner. We sampled 1,000,000 users and videos from the app, and treat these items as nodes. The edges in the generated bipartite graph represent that the user clicks the short videos. Each user has 64 features, and the target is to category these short videos into 253 different classes.

A.2 Decoupled Neural Message Passing

Note that the aggregate and update operations are inherently intertwined in Equation (1), i.e., each aggregate operation requires a neural layer to update the node's hidden state in order to generate a new message for the next step. Recently, some researches show that such entanglement could compromise performance on a range of benchmark tasks [12, 52, 64], and suggest separating GCN from the aggregation scheme. We reformulate these models into a single Decoupled Neural Message Passing (DNMP) framework: Neural prediction messages are first generated (with update function) for each node utilizing only that node's own features, and then aggregated using aggregate function.

$$\mathbf{h}_v^0 \leftarrow \mathsf{update}(\mathbf{x}_v), \ \mathbf{h}_v^t \leftarrow \mathsf{aggregate}\left(\left\{\mathbf{h}_u^{t-1}|u \in \mathcal{N}_v\right\}\right). \qquad (11)$$

[1]https://github.com/tkipf/gcn/tree/master/gcn/data
[2]http://lms.comp.nus.edu.sg/research/NUS-WIDE.html
[3]https://github.com/GraphSAINT/GraphSAINT
[4]https://kddcup2016.azurewebsites.net/
[5]https://docs.dgl.ai/en/0.4.x/api/python/data.html#coauthor-dataset
[6]https://github.com/snap-stanford/ogb

where x_v is the input feature of node v. Existing methods, such as PPNP [21], APPNP [21], AP-GCN [42] and etc., follows this decoupled MP. Taking APPNP as an example:

$$\mathsf{APPNP\text{-}update}(\mathbf{x_v}) = \sigma(W\mathbf{x}_v),$$

$$\mathsf{APPNP\text{-}aggregate}\left(\left\{\mathbf{h}_u^{t-1}|u \in \mathcal{N}_v\right\}\right) = \alpha\mathbf{h}_v^0 + (1-\alpha)\sum_{u \in \mathcal{N}_v}\frac{\mathbf{h}_u^{t-1}}{\sqrt{\tilde{d}_v\tilde{d}_u}},$$

where aggregate function adopts personalized PageRank with the restart probability $\alpha \in (0,1]$ controlling the locality.

A.3 More details about the compared baselines

The main characteristic of all baselines are listed as follows:

- **GCN** [20] produces node embedding vectors by truncating the Chebyshev polynomial to the first-order neighborhoods.
- **ResGCN** [20] adopts the residual connections between hidden layers to facilitate the training of deeper models by enabling the model to carry over information from the previous layer's input.
- **JK-Net** [57] proposes a new aggregation scheme for node representation learning that can adapt neighborhood ranges to nodes individually.
- **APPNP** [21] uses the relationship between GCN and PageRank to derive an improved propagation scheme based on personalized PageRank.
- **AP-GCN** [42] is a variation of GCN wherein each node selects automatically the number of propagation steps performed across the graph.
- **SGC** [52] reduces the excess complexity of GCN through successively removing non-linearities and collapsing weight matrices between consecutive layers.
- **SIGN** [12] is a sampling-free Graph Neural Network model that is able to easily scale to gigantic graphs while retaining enough expressive power.
- **GraphSAGE** [15] is an inductive framework that leverages node attribute information to efficiently generate representations on previously unseen data.
- **GAT** [28] leverages masked self-attentional layers to address the shortcomings of prior GNNs based on graph convolutions or their approximations, and enables specifying different weights to different nodes in a neighborhood.
- **S²GC** [68]: S²GC uses a modified Markov Diffusion Kernel to derive a variant of GCN, and it can be used as a trade-off of low-pass and high-pass filter which captures the global and local contexts of each node.
- **FastGCN** [5] interprets graph convolutions as integral transforms of embedding functions under probability measures, and enhances GCN with importance sampling.
- **ClusterGCN** [7] designs the batches based on efficient graph clustering algorithms, and it proposes a stochastic multi-clustering framework to improve the convergence.
- **GBP** [6]: GBP utilizes a localized bidirectional propagation process to further improve SGC.

Table 6: Overview of the Graph Datasets

Dataset	#Nodes	#Features	#Edges	#Classes	#Train/Val/Test	Task type	Description
Cora	2,708	1,433	5,429	7	140/500/1000	Transductive	citation network
Citeseer	3,327	3,703	4,732	6	120/500/1000	Transductive	citation network
Pubmed	19,717	500	44,338	3	60/500/1000	Transductive	citation network
Amazon Computer	13,381	767	245,778	10	200/300/12881	Transductive	co-purchase graph
Amazon Photo	7,487	745	119,043	8	160/240/7,087	Transductive	co-purchase graph
ogbn-products	2,449,029	100	61,859,140	47	195922/489811/204126	Transductive	co-purchase network
Coauthor CS	18,333	6,805	81,894	15	300/450/17,583	Transductive	co-authorship graph
Coauthor Physics	34,493	8,415	247,962	5	100/150/34,243	Transductive	co-authorship graph
Flickr	89,250	500	899,756	7	44,625/22,312/22,312	Inductive	image network
Reddit	232,965	602	11,606,919	41	155,310/23,297/54,358	Inductive	social network
Industry	1,000,000	64	1,434,382	253	5,000/10,000/30,000	Transductive	user-video graph

A.4 Experiments setup

We use PyTorch [7] and DGL to implement the models, and we train them using Adam optimizer. To evaluate the scalability of Graph-SAGE, we implement GraphSAGE via DistDGL. Besides, we train each model 400 epochs and terminate the training process if the validation accuracy does not improve for 20 consecutive steps. Note that both GraphSAGE and JKNet have three aggregators, and we choose the concatenation and mean as their aggregator, respectively, since these two aggregators perform best in most datasets.

For GAT, the number of attention heads is fixed to 8. For Graph-SAGE, we use the results on Flickr and Reddit as reported in [15] and [60]. For ClusterGCN, we use the results on Reddit as reported in [7] and run our own implementation on Flickr. The hyperparameters are selected from random search. The random search was performed over the following search space: hidden size ∈ {8, 16, 32, 64, 128, 256, 512}, learning rate ∈ {1e-3, 5e-3, 1e-2, 5e-2, 1e-1, 2e-1}, dropout rate ∈ {0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.85, 0.9]}, regularization strength ∈ {1e-4, 5e-4,1e-3, 5e-3, 1e-2, 5e-2, 1e-1}. Note that both Res-GCN and JK-Net will degrade into GCN if they have only two layers, so we set their aggregation steps ∈ [3,20] in all datasets.

A.5 Experiment Environment and Reproduction Instructions

The experiments are implemented on 4 machines with 14 Intel(R) Xeon(R) CPUs (Gold 5120 @ 2.20GHz) and four NVIDIA TITAN RTX GPUs. The code is written in Python 3.6, and the multi-objective algorithm is implemented based on OpenBox [25]. We use Pytorch 1.7.1 on CUDA 10.1 to train the model on GPU.

---

[7]https://github.com/pytorch