PaSca: a New Paradigm and System to Build Scalable Graph Neural Networks

Anonymous Author(s)

ABSTRACT

In recent studies, 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 GNN, current researches focus on specific GNN design, rather than the general *design space*, which limits the discovery of potential scalable GNN models. Moreover, existing solutions cannot support extensive exploration over the design space for scalable GNNs. 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 models discovered by our system achieve consistent performance among competitive baselines. Concretely, PaSca-V2 and V3 outperform the state-of-the-art method JK-Net by 0.2% and 0.4% in terms of predictive accuracy on our large industry dataset while achieving up to 56.6× and 28.3× training speedups, respectively.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

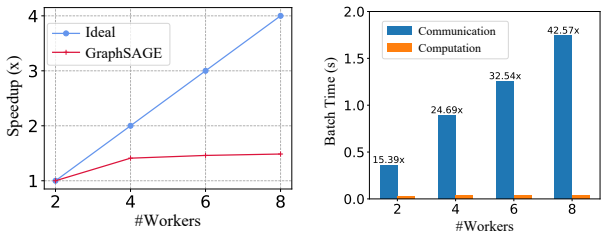
Scalable Graph Neural Network; Message Passing; Neural Architecture Search

ACM Reference Format:

Anonymous Author(s). 2022. PaSca: a New Paradigm and System to Build Scalable Graph Neural Networks. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3448016.3457325

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3457325



(a) Speedup (b) Bottleneck

Figure 1: The speedup and bottleneck of a two-layer GraphSAGE along with the increased workers on Reddit dataset (batch size = 8192).

1 INTRODUCTION

Graph neural networks (GNNs) [39] have become the state-of-the-art methods in many supervised and semi-supervised graph representation learning scenarios such as node classification [11, 13, 15, 17, 27, 40], link prediction [3, 45, 46], recommendation [1, 27, 41], and knowledge graphs [19, 26, 36, 37]. The majority of these GNNs can be described in terms of the neural message passing (NMP) framework [13], which is based on the core idea of recursive neighborhood aggregation. Specifically, during each iteration, the representation of each node is updated (with neural networks) based on messages received from its neighbors. Despite their success, existing GNN algorithms suffer from two scalability issues:

(1) Data Scalability. Most researches on GNNs are tested on small benchmark graphs, while graphs usually contain billions of nodes and edges in practice. However, we find that neural message passing does not scale well to large graphs since they typically need to perform a recursive neighborhood expansion to gather neural messages repeatedly. For large graphs that can not be entirely stored on each worker's local storage [25, 44, 50], gathering the required neighborhood neural messages leads to massive data communication cost [23, 33, 47]. To demonstrate this issue, we utilize distributed training functions provided by DGL [2] to test the scalability of GraphSAGE [13]. We partition the Reddit dataset across multiple machines and treat each GPU as a worker. 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 significantly limited even when the mini-batch training and graph sampling method are adopted. Note that the speedup is calculated relative to the runtime of two workers. 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 multi-scale messages.

(2) Model Scalability. Moreover, we find that GNNs under NMP principle cannot scale to large aggregation steps, as shown in Figure 2. This is because the existing NMP schemes are inflexible since they are restricted to a fixed-hop neighborhood and insensitive

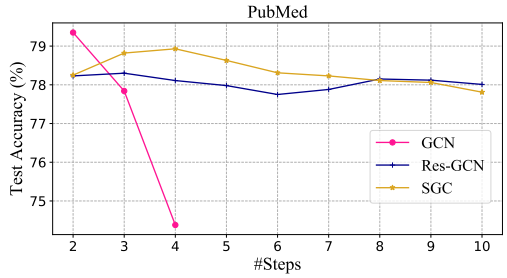


Figure 2: Test accuracy of different models along with the increased aggregation steps in PubMed.

to the actual demands of different nodes. It leads to two limitations: (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 [5, 20, 40]). As a result, most state-of-the-art GNN models are limited by model scale (e.g., usually two layers), preventing existing methods from unleashing their full potential.

Current Practice. While there has been an emerging interest in specific architectural designs to alleviate the above scalability issues, current researches only focus on *specific GNN design*, rather than the *design space*, which limits the discovery of potential scalable GNN models. In addition, existing solutions cannot support extensive exploration over design space for scalable GNNs, which is also a major motivation that inspires our work.

Our Contributions. 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:

(1) *Paradigm and Design Space*. We deconstruct GNN from the message passing mechanism and design the Scalable Graph Neural Architecture Paradigm (SGAP). SGAP introduces three decoupled stages to construct scalable GNNs with the following operation abstractions : (1) `graph_aggregator` captures structural information via graph aggregation operations, (2) `message_aggregator` combines different levels of structural information, and (3) `message_updater` generates the prediction based on multi-scale features. SGAP separates the update of neural network from message passing and leverages multiple messages over different levels of localities to improve performance. Following the SGAP paradigm, we 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 [38], SIGN [10], S^2GC [49] and GBP [7] are special instances in our design space. Empirical studies further show the superior scalability of architectures in the design space on training workers.

(2) *System Engine*. We implement an auto-search system engine that automates the search procedure for well-performing scalable GNN architectures instead of manual design. Models discovered using our design space achieve state-of-the-art performance. In addition, the engine can find Pareto-optimal solutions given multiple criteria (e.g., predictive performance, inference time, resource consumption), allowing a designer to select the best Pareto-optimal solution based on specific requirements.

(3) *State-of-the-art Performance*. Based on PaSca, we discover three novel scalable GNN instances, named PaSca-V1, V2, and V3, from the proposed design space for different accuracy-efficiency requirements. Extensive experiments on ten datasets demonstrate the superior transfer ability and training efficiency of PaSca models among competitive baselines. Concretely, PaSca-V2 and V3 outperform the state-of-the-art method JK-Net by 0.2% and 0.4% in terms of predictive accuracy on our large industry dataset, while achieving up to 56.6× and 28.3× training speedups, respectively.

2 PRELIMINARY

2.1 GNNs and Message-Passing

Considering a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V}$, edges $\mathcal{E}$ and features for all nodes $\mathbf{x}_v \in \mathbb{R}^d, \forall v \in \mathcal{V}$, many proposed GNN models can be analyzed using the message passing (MP) framework.

Neural Message Passing (NMP). This NMP framework is widely adopted by mainstream GNNs, like GCN [15], GraphSAGE [13], GAT [34], and GraphSAINT [43], where each layer adopts a neighborhood and an update function. At timestep t, a message vector $\mathbf{m}_v^t$ for node $v \in \mathcal{V}$ 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, which is:

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

Messages are passed for K timesteps so that the steps of message passing correspond to the network depth. Taking the vanilla GCN [15] as an example, we have:

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

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

where $\tilde{d}_v$ is the degree of node v obtained from the adjacency matrix with self-connections $\tilde{A} = I + A$.

Decoupled Neural Message Passing (DNMP). 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 [10, 38], and suggest separating GCN from the aggregation scheme. We reformulate these models into a single decoupled NMP 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.

$$\mathbf{h}_v^0 \leftarrow \texttt{update}(\mathbf{x}_v),\ \mathbf{h}_v^t \leftarrow \texttt{aggregate}\left(\{\mathbf{h}_u^{t-1} | u \in \mathcal{N}_v\}\right). \quad (2)$$

where x_v is the input feature of node v. Existing methods, such as PPNP [17], APPNP [17], AP-GCN [32] and etc., follows this decoupled MP. Taking APPNP as an example:

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

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

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

2.2 Related Works

Scalable GNN Instances. Following SGC [38], 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 [14]. Concretely, SIGN [30] proposes to concatenate different iterations of aggregated feature messages, while S^2GC [49] proposes a simple spectral graph convolution to average them. In addition, GBP [7] 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. Recently, graph neural architecture search has been proposed to solve the labor-intensive problem of GNN architecture design. Auto-GNN [48] and GraphNAS [12] are early approaches that apply reinforcement learning with RNN controllers on a fixed search space. You et al. [42] define a similarity metric for tasks and search for the best transferable architectures across tasks via random search. Based on DARTS [24], GNAS [4] proposes the differentiable searching strategy to search for GNNs with optimal message-passing depth. DSS [22] also adopts the differentiable strategy but optimizes over a dynamically updated search space. While previous studies own individual characteristics, PaSca differs from them in the following two aspects: 1) PaSca targets at searching for scalable architectures instead of general architectures to pursue efficiency on large-scale graphs; 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 SYSTEM

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 framework, and present an auto-search system engine to automate the exploration of design space.

3.1 SGAP Paradigm

Our PaSca system introduce 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 aggregation step t from 1 to K_{pre}, and at each step t, we use the `graph_aggregator` to aggregate the message vector collected from the neighbors $\mathcal{N}_v$:

$$\mathbf{m}_v^t \leftarrow \texttt{graph\_aggregator}\left(\{\mathbf{m}_u^{t-1} | u \in \mathcal{N}_v\}\right), \tag{3}$$

where $\mathbf{m}_v^0 = \mathbf{x}_v$. The messages are passed for K_{pre} steps, and $\mathbf{m}_v^t$ at timestep t can gather the neighborhood information from nodes that are t-hop away (lines 4-7). 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

Algorithm 1: An example of scalable graph neural architectures following SGAP.

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, aggregation depth K_{pre}, K_{post}, features $\mathbf{x}_v$ for each node $v \in \mathcal{V}$.
Output: Prediction message $\mathbf{m}_v^{K_{post}}$ for each node $v \in \mathcal{V}$.

1 Initialize message set $\mathcal{M}_v = \{\mathbf{x}_v\}$ for each node $v \in \mathcal{V}$;
2 **Stage 1: Pre-processing**
3 Initialize feature message $\mathbf{m}_v^0 = \mathbf{x}_v$ for each node $v \in \mathcal{V}$;
4 **for** $1 \le t \le K_{pre}$ **do**
5 **for** $v \in \mathcal{V}$ **do**
6 $\mathbf{m}_v^t \leftarrow \texttt{graph\_aggregator}(\mathbf{m}_{\mathcal{N}_v}^{t-1})$;
7 $\mathcal{M}_v = \mathcal{M}_v \cup \{\mathbf{m}_v^t\}$;
8 $\mathbf{c}_v \leftarrow$`message_aggregator`$(\mathcal{M}_v)$;
9 **Stage 2: Model-training**
10 **for** $v \in \mathcal{V}$ **do**
11 $\mathbf{h}_v \leftarrow$`message_updater`$(\mathbf{c}_v)$;
12 **Stage 3: Post-processing**
13 Initialize feature message $\mathbf{m}_v^0 = \mathbf{h}_v$ for each node $v \in \mathcal{V}$;
14 **for** $1 \le t \le K_{post}$ **do**
15 **for** $v \in \mathcal{V}$ **do**
16 $\mathbf{m}_v^t \leftarrow \texttt{graph\_aggregator}(\mathbf{m}_{\mathcal{N}_v}^{t-1})$;

message vector $\mathbf{c}_v$ for each node v (line 8) as:

$$\mathbf{c}_v \leftarrow \texttt{message\_aggregator}(\mathcal{M}_v). \tag{4}$$

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 $\mathbf{m}_v^{K_{pre}}$.

Model-training. Given the combined features and ground-truth labels, the `message_updater` is used to learn the class distribution of all nodes, i.e., the soft predictions (softmax outputs) predicted by the updater (line 10-11). Specifically, PaSca learns node representation $\mathbf{h}_v$ by applying fully-connected layers to the combined message vector $\mathbf{c}_v$:

$$\mathbf{h}_v \leftarrow \texttt{message\_updater}(\mathbf{c}_v). \tag{5}$$

Post-processing. Motivated by Label Propagation [35], we regard the soft predictions as new features and then use the `graph_aggregator` again at each step to aggregate the adjacent node predictions and make the final prediction (lines 14-16) as:

$$\mathbf{m}_v^t \leftarrow \texttt{graph\_aggregator}\left(\{\mathbf{m}_u^{t-1} | u \in \mathcal{N}_v\}\right), \tag{6}$$

where $\mathbf{m}_v^0 = \mathbf{h}_v$. We summarize the key differences between SGAP and existing GNN message passing (MP) framework (Section 2.1) as follows,

(1) Message type. To address the scalability challenge, the message type in our abstraction is different from the previous MP framework. We propose to pass node *feature/prediction messages* instead of neural messages by making message aggregation independent of the update operation. Concretely, to perform the aggregate function for the next step, MP in existing GNNs updates the hidden state $\mathbf{h}_v^t$ by applying the message vector $\mathbf{m}_v^t$ with neural networks. The decoupled MP framework also needs to update the hidden state $\mathbf{h}_v^t$ with neural networks first in order to get the neural

message for the following multi-step aggregate procedure. 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) Message scale. The existing MP framework only utilizes the last message vector $\mathbf{m}_v^{K_{pre}}$ to compute the final hidden state $\mathbf{h}_v^{K_{pre}}$. Motivated by the observation in Section 2.2, SGAP assumes that the optimal neighborhood expansion size should not be the same for each node v, and thus retaining 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 with `message_aggregator`, such that we could balance the preservation of information from both local and extended (multi-hop) neighborhoods for a given node.

3.2 Proposed Design Space

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 depth 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 timestep. 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) [15]. 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:

$$\mathbf{m}_v^t = \sum_{u \in \mathcal{N}_v} \frac{1}{\tilde{d}_u} \mathbf{m}_u^{t-1}. \tag{7}$$

Personalized PageRank (PPR) [16]. 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 [16]:

$$\mathbf{m}_v^t = \alpha \mathbf{m}_v^0 + (1-\alpha) \sum_{u \in \mathcal{N}_v} \frac{1}{\sqrt{\tilde{d}_v \tilde{d}_u}} \mathbf{m}_u^{t-1}, \tag{8}$$

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.

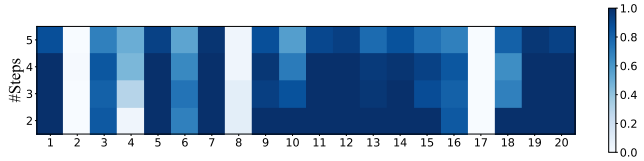


Figure 3: The influence of aggregation steps on 20 randomly sampled nodes on Citeseer dataset. The X-axis is the node id and Y-axis is the aggregation steps (number of layers in GCN). The color from white to blue represents the ratio of being predicted correctly in 50 different runs.

Triangle-induced adjacency (Triangle. IA) [28]. 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 metrix 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:

$$\mathbf{m}_v^t = \sum_{u \in \mathcal{N}_v} \frac{1}{d_v^{tri}} \mathbf{m}_u^{t-1}. \tag{9}$$

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 aggregators per node into a single vector, such that the subsequent model learns from the multi-scale neighborhood of a given node. We summarize the 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,

$$c_{msg} \leftarrow \oplus_{\mathbf{m}_v^i \in M_v} w_i f(\mathbf{m}_v^i), \tag{10}$$

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 [7]. Compared with pooling operators, though the concatenating operator keeps all the input message information, the dimension of its outputs increases as K_{pre} grows, leading to additional computational cost in the downstream updater.

Adpative aggregators. We observe that messages of different hops make different contributions to the final performance. As shown in Figure 3, we apply standard GCN with different layers to conduct node classification on the Cora dataset. We observe that most of the nodes are well classified with two steps, and as a result, most state-of-the-art GNN models are designed 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.

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(α = 0.1), PPR(α = 0.2), PPR(α = 0.3), Triangle. IA}	Categorical
	Message aggregators (MA)	{None, Mean, Max, Concatenate, Weighted, Adaptive}	Categorical
Model training	Transformation steps (K_{trans})	[1, 10]	Integer
Post-processing	Aggregation steps (K_{post})	[0, 10]	Integer
	Graph aggregators (GA_{post})	{Aug.NA, PPR(α = 0.1), PPR(α = 0.2), PPR(α = 0.3), Triangle. IA}	Categorical

$$\mathbf{c}_{\text{msg}} \leftarrow \sum_{\mathbf{m}_v^i \in M_v} w_i \mathbf{m}_v^i, \;\; w_i = \sigma(\mathbf{s}\mathbf{m}_v^i), \tag{11}$$

where $\mathbf{s}$ is a trainable vector shared by all nodes to generate gating scores, and σ denotes the sigmoid function. By utilizing the adaptive message aggregator, the model is capable of balancing the messages from the multi-scale neighborhoods for each node at the expense of training extra parameters.

3.2.3 Specific Instances in Design Space. Recent popular data-scalable GNN models, such as SGC [38], SIGN [10], S^2GC [49] and GBP [7], can be considered as specific instances in our defined design space. As shown in Table 2, most current scalable GNNs ignore the post-processing stages, which can boost the model performance in most circumstances in our experiments. Besides, various scalable GNNs can be obtained by using different settings. 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`. In addition, we decouple MLP training and label propagation in APPNP [17] into two individual processes and get a new scalable model PaSca-APPNP.

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	/
S^2GC	PPR	Mean	1	/
GBP	Aug.NA	Weighted	≥ 2	/
PaSca-APPNP	/	/	≥ 2	PPR

3.3 Auto-search System Engine

To effectively explore the large design space, we design an auto-search system engine to automate the search procedure of scalable GNN architecture instead of manual design. Figure 4 shows the overview of our proposed auto-search system engine to explore GNN designs. It consists of two components: the search engine and the evaluation engine. The entire 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 according to its built-in optimization algorithm and sends it to the evaluation engine; 2) The evaluation engine evaluates the corresponding architecture and updates the suggestion server with the observed objective values. More details about each component are introduced in the following.

Search Engine. While prior researches on scalable GNNs [7, 10] 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.

Evaluation Engine. Different from the existing GNNs, the evaluation process of PaSca is separated into two processing stages and one training stage as shown in Figure 4: 1) We pre-compute the feature messages for each node over the graph; 2) We combine the messages and train the model parameters with SGD; 3) We post-compute 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.

Message processing includes pre-computing and post-computing. They share the same pipeline but take different messages as inputs (features for pre-computing and predictions for post-computing). 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. Since we compute

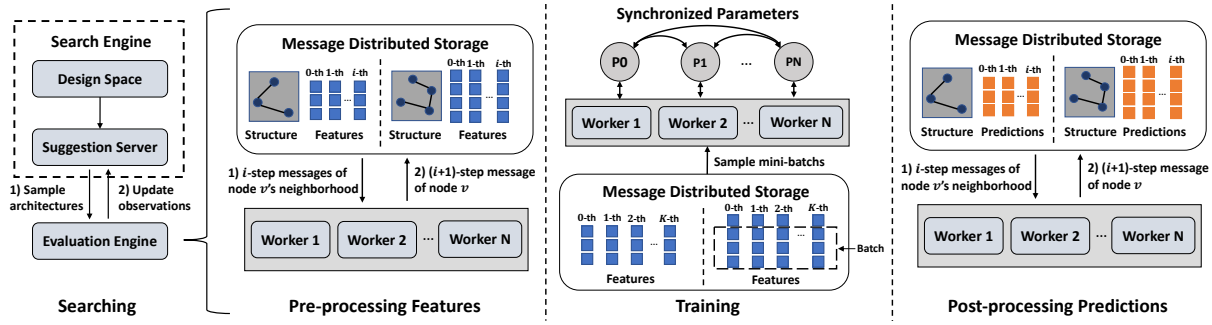


Figure 4: The workflow of PaSca, consisting of message pre-compute stage and message aggregation/update stage.

the message vectors for each node in parallel, our implementation could scale to large graphs and significantly reduce the runtime.

For *architecture training*, 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.

Table 3: Dataset overview. (T and I represents transductive and inductive, respectively)

Dataset	#Nodes	#Features	#Edges	#Classes	Task
Cora	2,708	1,433	5,429	7	T
Citeseer	3,327	3,703	4,732	6	T
PubMed	19,717	500	44,338	3	T
Amazon Computer	13,381	767	245,778	10	T
Amazon Photo	7,487	745	119,043	8	T
Coauthor CS	18,333	6,805	81,894	15	T
Coauthor Physics	34,493	8,415	247,962	5	T
ogbn-products	2,449,029	100	61,859,140	47	T
Flickr	89,250	500	899,756	7	I
Reddit	232,965	602	11,606,919	41	I
Industry	1,000,000	64	1,434,382	253	T

4 EXPERIMENTS

4.1 Experimental Settings

Datasets. We conduct the experiments on public partitioned datasets, including three citation networks (Citeseer, Cora, and PubMed) in [15], two social networks (Flickr and Reddit) in [43], four co-authorship graphs (Amazon and Coauthor) in [29], the co-purchasing network (ogbn-products) in [14] and one short-form video recommendation graph (Industry) from our industrial cooperative enterprise. Table 3 provides the overview of the 11 datasets.

Parameters. For each baseline, we use random search or follow the original papers to get the optimal hyperparameters. To eliminate random factors, we run each method 20 times and report the mean and variance of the performance. We provide more details for reproduction at Anonymous Github [1].

[1]https://anonymous.4open.science/r/PASCA-CB24

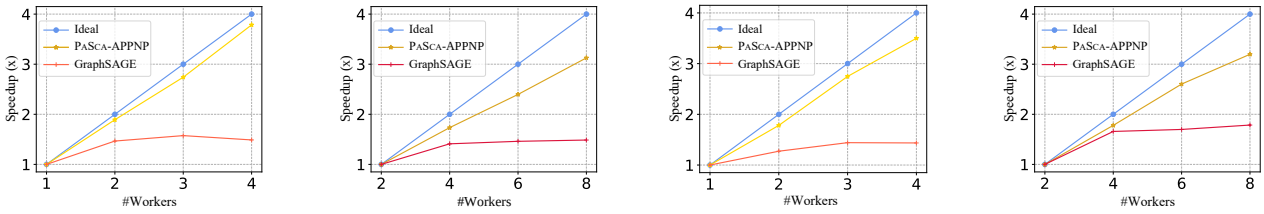
Environment. We run our experiments on four machines, each with 14 Intel(R) Xeon(R) CPUs (Gold 5120 @ 2.20GHz) and 4 NVIDIA TITAN RTX GPUs. The codes are written in Python 3.6 with Pytorch 1.7.1, and the multi-objective algorithm is implemented based on OpenBox [21].

Baselines. In the transductive settings, we compare the searched scalable GNNs with GCN [15], GAT [34], JK-Net [40], Res-GCN [15], APPNP [18], AP-GCN [32], SGC [38], SIGN [31], S^2GC [49] and GBP [7]), which are SOTA models of different message passing types. In the inductive settings, the compared baselines are GraphSAGE [13], FastGCN [6], ClusterGCN [8] and GraphSAINT [43].

In the following, we first analyze the superiority of our proposed design space and then introduce the searched PaSca models via multiple-objective optimization. Finally, we evaluate the transfer ability, training efficiency, and model scalability of PaSca models compared with competitive state-of-the-art baselines.

4.2 Design Space Analysis

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, and the results are shown in Figure 5. We run the methods in both stand-alone and distributed scenarios and then measure their corresponding speedups. The batch size is set to 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. For GraphSAGE, since it requires aggregating the neighborhood nodes during training, it meets the I/O bottleneck when transmitting a large number of required neural messages. Thus, the speedup of GraphSAGE increases slowly as the number of workers grows. The speedup of GraphSAGE 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 behaves close to the ideal circumstance in both scenarios, indicating the superiority of our proposed design space.



(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.

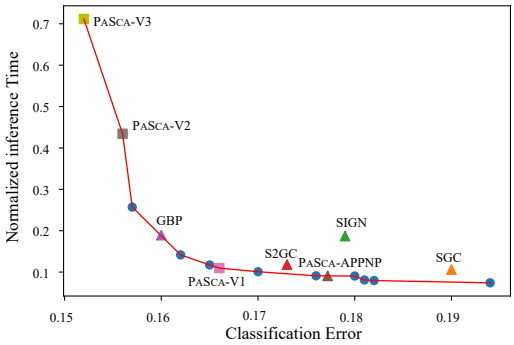


Figure 6: Pareto Front found on Cora.

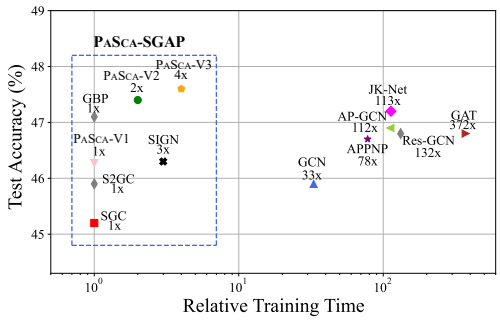


Figure 7: Test accuracy over training time on Industry.

Table 4: 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

4.3 Searching Results

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, falls on the Pareto Front, which indicates the superior design of "Weighted" message aggregator and "PPR" graph aggregator. In addition, we select three architectures from the Pareto Front as PaSca-V1 to V3 for different accuracy-efficiency requirements. The corresponding parameters of each architecture are shown in Table 4. Among the three architectures, PaSca-V1 Pareto-dominates the other baselines except GBP, and PaSca-V3 is the architecture with the best predictive performance found by the search engine.

4.4 Performance-Efficiency Analysis

To test the transfer ability and training efficiency of PaSca models, we further evaluate them on more datasets compared with competitive baselines. The results are summarized in Table 5 and 6.

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 6 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.

4.5 Model Scalability

We observe that both PaSca-V2 and V3 found by the search engine contain the "Adaptive" message aggregator. In this subsection, we aim to explain the advantage of adaptive message aggregator in the perspective of model scalability on message passing steps. We plot

Table 5: 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^2GC	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 6: 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**

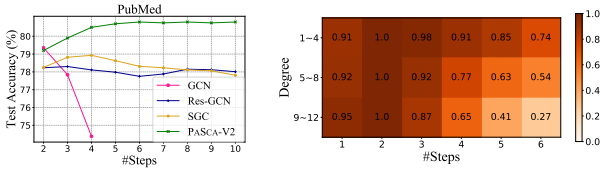


Figure 8: Left: Test accuracy of different models along with the increased aggregation steps on PubMed. Right: PaSca-V2's average gating weights of graph messages of different steps on 60 randomly selected nodes from PubMed.

the changes of model performance along with the message passing steps in the left subfigure of Figure 8. For 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. SGC alleviates this problem by removing the non-linear transformation, and Res-GCN carries information from the previous step by introducing the residual connections. However, their performance still degrades as 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 non-decreasing across steps, which indicates that PaSca scales to large depths. This is because the adaptive message aggregator in PaSca-V2 is able to adaptively and effectively combine multi-scale neighborhood messages for each node.

To demonstrate this, the right subfigure of Figure 8 shows sys-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 2-layer methods 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.

5 CONCLUSION

In this paper, we proposed PaSca, a new paradigm and system which 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 models generated by PaSca outperform the SOTA GNNs in terms of performance, efficiency, and scalability. PaSca can help GNN researchers understand design choices when developing new scalable GNN models, and can serve as a system to support extensive exploration over the design space for scalable GNNs.

REFERENCES

[1] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).

[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] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3438–3445.

[6] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[7] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable Graph Neural Networks via Bidirectional Propagation. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

[8] 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, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 257–266.

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

[10] 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*.

[11] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1416–1424.

[12] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2019. Graphnas: Graph neural architecture search with reinforcement learning. *arXiv preprint arXiv:1904.09981* (2019).

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.

[14] 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. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.).

[15] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[16] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Personalized Embedding Propagation: Combining Neural Networks on Graphs with Personalized PageRank. *CoRR* abs/1810.05997 (2018).

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

[18] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[19] Chung-Wei Lee, Wei Fang, Chih-Kuan Yeh, and Yu-Chiang Frank Wang. 2018. Multi-label zero-shot learning with structured knowledge graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1576–1585.

[20] 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.

[21] 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. *arXiv preprint arXiv:2106.00421* (2021).

[22] Yanxi Li, Zean Wen, Yunhe Wang, and Chang Xu. 2021. One-shot Graph Neural Architecture Search with Dynamic Search Space. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 8510–8517.

[23] Zhiqi Lin, Cheng Li, Youshan Miao, Yunxin Liu, and Yinlong Xu. 2020. PaGraph: Scaling GNN training on large graphs via computation-aware caching. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 401–415.

[24] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[25] Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. 2019. Neugraph: parallel deep neural network computation on large graphs. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*. 443–458.

[26] Kenneth Marino, Ruslan Salakhutdinov, and Abhinav Gupta. 2016. The more you know: Using knowledge graphs for image classification. *arXiv preprint arXiv:1612.04844* (2016).

[27] Federico Monti, Michael M Bronstein, and Xavier Bresson. 2017. Geometric matrix completion with recurrent multi-graph neural networks. *arXiv preprint arXiv:1704.06803* (2017).

[28] Federico Monti, Karl Otness, and Michael M. Bronstein. 2018. MotifNet: a motif-based Graph Convolutional Network for directed graphs. *CoRR* abs/1802.01572 (2018).

[29] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

[30] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. 2020. SIGN: Scalable Inception Graph Neural Networks. *arXiv preprint arXiv:2004.11198* (2020).

[31] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael M. Bronstein, and Federico Monti. 2020. SIGN: Scalable Inception Graph Neural Networks. *CoRR* abs/2004.11198 (2020).

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

[33] Alok Tripathy, Katherine Yelick, and Aydin Buluc. 2020. Reducing communication in graph neural network training. *arXiv preprint arXiv:2005.03300* (2020).

[34] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

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

[36] 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.

[37] Zhouxia Wang, Tianshui Chen, Jimmy Ren, Weihao Yu, Hui Cheng, and Liang Lin. 2018. Deep reasoning with knowledge graph for social relationship understanding. *arXiv preprint arXiv:1807.00504* (2018).

[38] 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.

[39] 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).

[40] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*. 5449–5458.

[41] 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.

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

[43] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[44] Dalong Zhang, Xin Huang, Ziqi Liu, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Zhiqiang Zhang, Lin Wang, Jun Zhou, and Yuan Qi. 2020. AGL: a scalable system for industrial-purpose graph machine learning. *arXiv preprint arXiv:2003.02454* (2020).

[45] Muhan Zhang and Yixin Chen. 2017. Weisfeiler-lehman neural machine for link prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 575–583.

[46] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *arXiv preprint arXiv:1802.09691* (2018).

[47] 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. *arXiv preprint arXiv:2010.05337* (2020).

[48] 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).

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

[50] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. 2019. Aligraph: A comprehensive graph neural network platform. *arXiv preprint arXiv:1902.08730* (2019).