FGNAS: FPGA-AWARE GRAPH NEURAL ARCHITECTURE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

The success of graph neural networks (GNNs) in the past years has aroused growing interest and effort in designing best models to handle graph-structured data. As the neural architecture search (NAS) technique has been witnessed to rival against human experts in discovering efficient network topology, recently, it has been applied to the field of graphic network engineering. However, such works on graphic NAS so far are purely software (SW) design and not considering hardware (HW) constraints at all, which often leads to sub-optimal system performance. To address this problem, we propose the first SW-HW co-design framework for automating the search and deployment of GNNs. Using FPGA as the target platform, our framework is able to perform the FPGA-aware graph neural architecture search (FGNAS). To evaluate our design, we experiment on benchmark datasets, namely Cora, CiteCeer, and PubMed, and the results show FGNAS has better capability in optimizing the accuracy of GNNs when their hardware implementation is specifically constrained.

1 INTRODUCTION

Graph neural networks (GNNs) are the state of the art in solving machine learning problems represented in graph forms, including social networking (Tan et al., 2019; Nurek & Michalski, 2019), molecular interaction (Huang et al., 2020; Spalević et al., 2020), and problems in Electronic Design Automation (EDA) (Ma et al., 2020; Ma et al., 2019), etc. As a result, GNN has attracted a great deal of research interest in deep learning community for both software (SW) (Wu et al., 2019; Li et al., 2015) and hardware (HW) (Wang et al., 2020; Zeng & Prasanna, 2020).

Similar to many other neural networks, the performance of GNN significantly depends on its neural architecture, and hence considerable effort has been put into tuning its computational components (Hamilton et al., 2017). Among the existing algorithms, message-passing has set the ground of spatial-based convolutional graph neural networks, from which most recent breakthough are derived (Gilmer et al., 2017). As the algorithmic variation increases, to identify better sub-structures of GNN tends to be substantially challenging due to the design space exponentially grows. On the other hand, however, the improvement of feature-extracting ability is still highly demanded.

Soon after being proposed by Zoph & Le (2016), neural architecture search has become a mainstream research topic of machine learning. It has been demonstrated NAS is promising to surpass the human experts and meanwhile liberate their laborious effort (Chen et al., 2018). Although the original NAS using reinforcement learning method suffers from timing inefficiency problem that following works strived to solve (Yan et al., 2019; Liu et al., 2019), it is well established thus adapted to be used for searching novel GNNs.

Quite lately, Gao et al. (2019) has designed the first graph NAS framework. Based on the state-of-art GNN methodology, Graph NAS has formulated the layered design space that is perferred to the controller. Besides, parameter sharing strategy is also adopted. Coincidentally, Zhou et al. (2019) has also used reinforcement learning to automate graph neural network design on similar search space but with split controllers. The search process is well guided in an incremental manner such that the sampling efficiency is boosted. Both of these works have improved the accuracy of GNN against existing hand-crafted networks, indicating NAS is the future solution for graph-based learning.

However, these works are only focusing on the neural architecture while the hardware implementation for GNNs (Geng et al., 2019) is equally important to the final performance. The hardware-aware NAS has been widely discussed for CNNs (Zhang et al., 2020; Wang et al., 2018). But, to our best knowledge, joint search of hardware and GNN architectures have not publicly reported. In this paper, we use Graph NAS with the hardware design objective and propose a software-hardware co-design framework. We employ FPGA as the vehicle for illustration and implementation of our methods. Specific hardware constraints are considered so quantization is adopted to compress the model. Under specific hardware constraints, we show our framework can successfully identify a solution of higher accuracy but using shorter time than random search and the traditional two-step tuning.

2 PROBLEM FORMULATION

The problem of jointly searching graph neural network architectures and hardware design can be formulated as the following. Given an architecture space $\mathcal{A}$, each sample $a \in \mathcal{A}$ characterizes a hardware space $\mathcal{H}(a)$. The objective is then to find the optimal architecture and hardware design pair $\langle a^*, h^* \rangle$ such that $a^* \in \mathcal{A}$ and $h \in \mathcal{H}(a^*)$. With the target dataset D_t for training and D_v for validation, the accuracy of a design can be measured as $acc_t(a, h)$ and $acc_v(a, h)$, respectively, while the hardware performance $hp(a, h)$ is independent of the data. As the neural architecture sample is parameterized by the weights w, we define the optimality of the design as

$$\begin{aligned} a^* &= \arg\max_{a \in \mathcal{A}} acc_v(a(w^*), h^*) \\ s.t. : w^* &= \arg\max_{w} acc_t(a(w), h^*) \end{aligned} \tag{1}$$

and at the same time

$$\begin{aligned} h^* &= \arg\max_{h \in \mathcal{H}(a^*)} hp(a^*, h) \\ s.t. : hp(a^*, h^*) &\geq spec \end{aligned} \tag{2}$$

where $spec$ is the hardware specification required to be satisfied by the design.

However, there is a problem with the above formulation that is challenging to implementation. In the case where the specification of hardware relate to multiple objectives, e.g. area and latency, the hardware performance is not a scalar and hence the optimization is ambiguous. In practice, the design is acceptable as long as the hardware constraints are met. In order to optimize the hardware design, one can set more and more strict constraints to the aspect of interest. Therefore, we relax the optimization of hardware performance to the hardware eligibility, and reformulate the problem as

$$\begin{aligned} a^* &= \arg\max_{a \in \mathcal{A}} acc_v(a(w^*), h) \\ s.t. : w^* &= \arg\max_{w} acc_t(a(w), h) \end{aligned} \tag{3}$$

and

$$\begin{aligned} &\exists h \in \mathcal{H}(a^*) \\ s.t. : hp(a^*, h) &\geq spec. \end{aligned} \tag{4}$$

It is worth mentioning when the hardware constraint has two and more dimensions, the $\geq$ symbol applies to every dimension.

In this work, we rely on the recurrent neural network to jointly optimize both the GNN architecture and its hardware design. As such, the reinforcement learning NAS framework is restructured to co-exploring the software and hardware spaces. Based on the above formulation, our framework aims to discover the best neural architectures which are guaranteed to be implementable.

3 FGNAS

In this section, we delve into the details of our FPGA-aware graph nerual architecture search (FGNAS) framework. As shown in Figure 1, there are three main components comprising FGNAS,

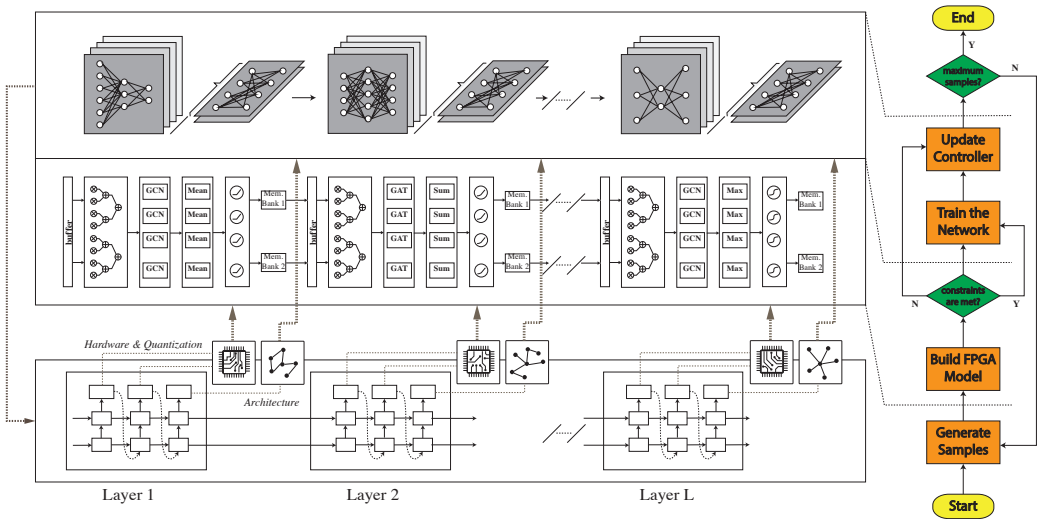

Figure 1: Overview of the proposed SW-HW co-design framework.

namely the controller, the FPGA model builder, and the gnn model trainer. For each layer of the child network, our controller generates the parameters of three types defining the network topology, hardware realization, and the precision. With each sample of the controller, a hardware model will be firstly constructed and evaluated against the predefined constraints. Since most samples may not be implementable, their training are circumvented and rewards assigned to be 0; otherwise the network will be built, trained and validated. Finally, when a mini-batch of samples are evaluated, the parameters of the controller will be updated once. The process terminates after a certain number of episodes.

3.1 Search Space

We divide the search space into two sub-spaces: architectural space and hardware space. For each layer of a GNN, the search spaces are completely the same so the same types of parameters are sampled. For illustration convenience, we divide the parameters of a single layer and describe them as follows.

3.1.1 Architectural Space

The architectural space contains the parameters that defines the operational mechanism of graph network. As the time of writing, the topologies of GNNs share message-passing computational flow characterized by graph-wise convolution, and only vary in the way embedded features are generated and combined. In consequence, we define the architecture space regarding the tuning of sub-structures.

Basically, three separate stages are cascaded in each layer: (1) the embedding form last layer are linearly converted; (2) messages between each connected pair of nodes are weighted; and (3) new features of neig'hbouring nodes are aggregated to produce new embedding. Following the three operations, five parameters are included in the architectural space.

- *Embedding Dimension.* The embedding represents the features of the nodes extracted by the hidden layers. A linear operation is applied to convert the previous embedding into another space of d dimensions.
- *Attention Type.* The attention type refers to how the messages between connected nodes are weighted. For the new temporary embedding $H_{i,j}^{k}$, a coefficient is firstly computed for weighting it during the aggregation phase.
- *Aggregation Type.* For all the incoming messages, there different ways in mixing them to produce the new features. The common methods are namely, taking the summation, mean, and maximum.

- *Number of Heads.* We apply multi-headed attention to the GNN architecture as it is commonly used to stablize the performance. Heads of the same message are concatenated for every layer except the last one where they are averaged to match the output dimension.
- *Activation Function.* The activation function can add nonlinearity to the embedding. Considering the hardware constraints, we include four options for nonlinearity: "relu", "tanh", "sigmoid", and "elu".

3.1.2 HARDWARE SPACE

The computation of GNN for inference are all parallelizable in terms of the features of the same embedding. As a large dimension would require exponentially complex computation, it is necessary to divide the vector-wise operation into sub-tasks. Therefore, we choose the size for grouping the features as a key parameter to scale the hardware.

Almost all the main tasks can be divided, and we summarize them into four cases:

1. For the embedding to transform from T_i to T_o features, two parameters t_i and t_o are used for grouping them separately.
2. The attention coefficients possibly also require linear operation but the output is a scalar, so we only divide the input by size of t_{attn}.
3. The aggregation is similar to the above case in that there is only one output. We also assign one parameter t_{aggr} for it.
4. Lastly, the nonlinearity requires one-to-one operation on the feature vector. As this is probably the most challenging operation for hardware, we also group the features into size of t_{act}.

In addition to the architectural and hardware space, we also consider the mixed-precision design which play important roles in both software and hardware performance. In this case, the quantization space also needs to be explored and details is discussed in Section 3.4.

3.2 ALGORITHM

Reinforcement learning is applied in our design as the searching backbone. As we have parameterized the design of both architecture and hardware and formatted these parameters by layer, one RNN can be employed to sample the parameters sequentially as actions from the respective list of options. For the sampled design, the hardware performance is analyzed using our FPGA model, under the constraints of resources and latency. Only if the sample hardware design satisfy the hardware specifications, will the software design be trained and tested on the dataset. The reward for the sample $\langle a, h \rangle$ is then

$$R(a,h) = \begin{cases} 0, & hp(a,h) < spec \\ acc_v(a,h), & \text{otherwise} \end{cases} \tag{5}$$

This way, the training can be circumvented as possible and the search can be faster than pure NAS.

Once the reward is obtained, the parameter θ of the controller is updated following the policy gradient rule (Williams, 1992):

$$\nabla J(\theta) = \frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} \gamma^{T-t} \nabla_\theta \log \pi_\theta(a_t | a_{(t-1):1})(R_k - b) \tag{6}$$

where $J(\theta)$ is the expected reward at the initial step.

The controller is configured as the following. The number of steps T equals the total number of parameters to be sampled; the batch size for updating θ is $m = 5$ episodes; the reward is not discounted so $\gamma = 1$; and baseline b is the exponential moving average of the reward with a decaying factor of 0.9.

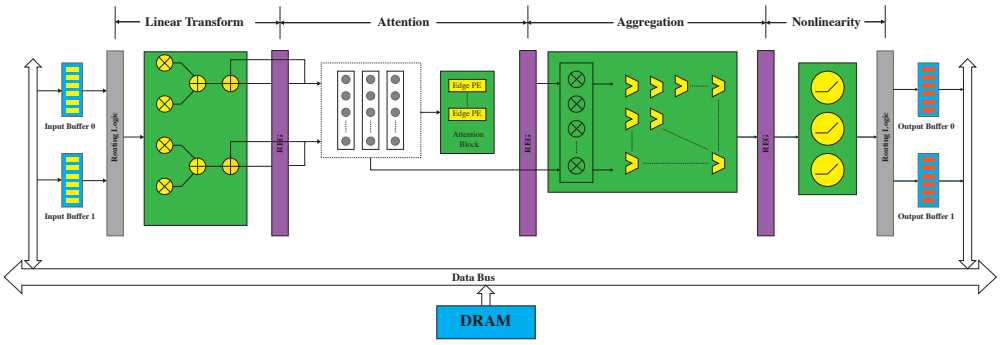

Figure 2: Block diagram of the FPGA model.

3.3 FPGA Modeling

We adopt a generic FPGA design model that is widely used for CNN acclerators (Zhang et al., 2015). Figure 2 illustrates the block diagram of the hardware segment for one layer. For each layer four stages are pipelined consisting of the linear transform, attention coefficient computation, aggregation, and nonlinear operation. The messages in-between consecutive stages are registered. Two buffers are employed to resolve the read/write conflict by alternately accessing the main memory and serving the computational units. As mentioned above, this model is fully scalable in the dimension of the embedded features baded on the parameteres defined.

3.4 Mixed Precision

We also consider the mixed-precision scenario in our design where data are quantized using different bit width. Like the other parameters, quantization parameters are also arranged by layer so data in the same layer share the same format. As the methods for quantizing is plentiful and have significant impact on the model accuracy, we avoid the variation of them and simply adopt the post-training quantization (PTQ) and linear quantization as follows.

Given the quantization interval Δ and range bounded by B_{min} and B_{max}, the quantization of real number x is

$$\hat{x} = clip(\lfloor x/\Delta \rceil \times \Delta, B_{min}, B_{max}), \tag{7}$$

where $\lfloor \rceil$ is rounding to integers. For the fixed-point format with sign, Δ, B_{min} and B_{max} are determined by the number of bits allocated to the integral (bi) and fractional (bf) part as

$$\Delta = 2^{-bf}, B_{min} = -2^{bi}, B_{max} = 2^{bi} - \Delta. \tag{8}$$

Consequently, in the mixed-precision design, four parameters are added to the search space namely wi, wf for the weights and ai and af for the activation. With the mixed precision, the hardware space exponentially increases, and the components in our FPGA model requires to be configured by bitwidth. We rely on the HLS tool of Xilinx to synthesize all configurations to profile the sizes and latency information. The synthesis result of sample operational units are shown in the supplimental material. It is noted the impact of quantization on hardware significantly vary among operators.

4 Experiment

In this section we test the performance of FGNAS on holdout graph datasets of node classification utility. To study the search efficiency, both of the test accuracy and searching time are evaluated and compared. The experiments are carried out using single Nvidia 1080Ti graphic processing unit (GPU), and Intel 8700K CPU. There is no dedication for FPGA chips, but we use Xinlix devices for reference. We assume the clock rate is 100 MHz throughout all the experiments. It is noted that since we constraints the hardware, comparing the accuracy to the state-of-art networks are not quite sensible and instead we evaluate the searching efficiency against baseline methods.

Table 1: Design space explored by our framework and the actual values used in the experiment.

Space	Parameter	Symbol	Value
Architecture	Embedding Dimension	d	4, 8, 12, 16, 32, 64
	Attention Type	$attn$	"constant", "gat", "gcn"
	Aggregation Type	$aggr$	"add", "max", "mean"
	Number of Heads	k	1, 2, 4, 8, 16
	Activation Function	act	"relu", "tanh", "sigmoid", "elu"
Hardware	Linear Group Size	t_{in}/t_{out}	1, 2, 3, 4, 5
	Attention Group Size	t_{attn}	1, 2, 4, 8
	Aggregation Group Size	t_{aggr}	1, 2, 4, 8
	Activation Group Size	t_{act}	1, 2, 3, 4, 5
	Integer Bit Width	ai/wi	1, 2, 3
	Fraction Bit Width	af/wf	0, 1, 2, 3, 4, 5, 6

Table 2: Basic information on the statistics of the datasets and our configuration in usage.

	Dataset	Cora	CiteSeer	PubMed
Statistics	# Training Nodes	140	120	60
	# Validation Nodes	500	500	500
	# Testing Nodes	1000	1000	1000
	# Input Features	1433	3703	500
	# Classes	7	6	3
training configuration	learning rate	0.01	0.01	0.01
	weight decay	0.0005	0.0005	0.001
Hardware Specification	Latency (ms)	0.8/0.9/1.0	0.8/0.9/1.0	7/8/9
	#LUT/#FF	10k/100k	10k/100k	10k/100k
	DSP	10/100	10/100	100/1000

4.1 DATASET

Three datasets are used for benchmarking the performance on transductive learning, namely Cora, CiteSeer, and PubMed. The statistics and training configuration is listed in Table 2. The setting for training on these datasets follows that of Zhou et al. (2019). Since the volumns and complexity of the datasets vary largely, the hardware of the search is constrained differently and accordingly.

4.2 BASELINE METHOD

To evaluate the ability and efficiency of FGNAS, two methods are considered as baseline and experimented in parallel with our method.

Random Search. We perform a random search approach as the baseline of search efficiency. The random search results can reflect the distribution of candidate solutions in specific design space. For certain data and hardware constraints, the random search can render decent result already.

Separate Search. The traditional method of two-phase design philosophy cannot fully explore the design space joined by hardware and architectural subspaces. In this philosophy, a fixed pure network architecture is firstly selected (by handcraft or automation), and afterwards a hardware design is customized for this specific architecture. Therefore, it explores only a fraction of the design space containing every combination of architecture-hardware pair.

To show the advantage of our co-design method over the separate design, we follow the above pipeline and partially use our framework to perform a pure architecture search followed by a pure hardware search based on the best network found.

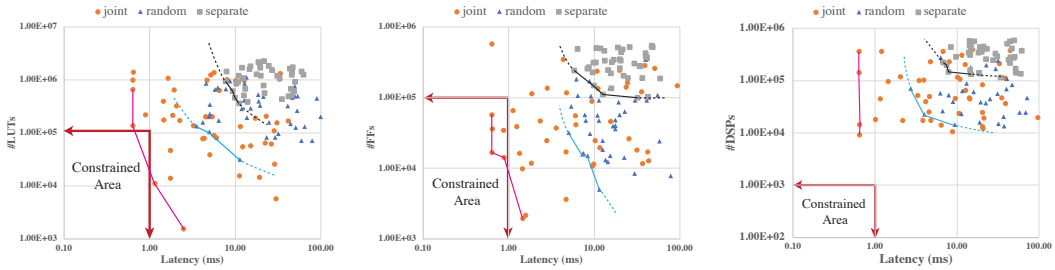

Figure 3: Distribution of searched samples on Cora. Our joint search methods explores the region that closest to the constrained area.

Table 3: Performance of the proposed joint search framework under different hardware constraints. Best test accuracy and search time on Cora are used to compare against the baselines.

Constraints			ours		random search		separate search	
latency (ms)	#LUT/#FF	#DSP	Acc.	Time (h)	Acc.	Time (h)	Acc.	Time (h)
0.8	10,000	10	66.2%	0.56	61.8%	1.12	62.6%	1.87
		100	62.9%	0.87	60.0%	1.22	63.8%	1.65
	100,000	10	67.8%	0.88	62.9%	1.12	64.6%	1.55
		100	68.7%	0.89	64.0%	0.95	68.5%	1.40
0.9	10,000	10	68.1%	0.69	68.0%	1.19	66.0%	1.32
		100	69.2%	1.17	68.9%	1.20	69.0%	1.50
	100,000	10	68.8%	0.99	69.0%	1.44	68.0%	1.69
		100	70.2%	0.88	69.5%	1.44	69.6%	1.70
1.0	10,000	10	68.1%	0.72	67.8%	1.23	66.0%	1.38
		100	70.1%	1.33	69.0%	1.40	69.9%	1.44
	100,000	10	68.8%	1.19	69.2%	1.66	69.0%	1.77
		100	71.5%	1.48	69.9%	1.55	69.9%	1.60

4.3 Searching Details

The actual search space used throughput the experiments are shown in Table 1. During the search, the controller is updated with ordinary SGD algorithm and a constant learning rate of 0.1. When a child network is sampled and hardware verified, it will be trained using Adam optimizer for 200 epochs. The validation is performed after every epoch, from which the highest will be taken as the reward to the controller. By rule of thumb, we set the depth of the child networks as two layers.

The searching stops after sampling 2000 episodes. With hardware constraints, however, most samples in both joint and random search may not be valid so the training can be saved. Consequently for fair comparison, we use the total number of *trained* samples to guide the random search such that the GPU hours would be on the same scale. In the case of separate search, the GPU time is completely defined by the episode quantity and we set 200 as for the architecture search and 800 for the hardware search. Each experiment includes 5 runs, and the one with the highest test accuracy is taken for evaluation. With the selected run, we report the accuracy of both the best sample as well as the top-10 samples averaged.

4.4 Performance

We test the seaching efficiency of our method across variational hardware parameters in latency, number of LUTs/FFs and number DSPs. The result on Cora is shown in Table 3. In general, the joint search ahieves the best accuracy and shortes searching time while there exists variance.

Table 4: The best accuacry result on differnt datasets.

Dataset	Latency	#LUT/#FF	#DSP	Ours	Random Search	Separate Search
Cora	1 ms	100k	100	71.5%	69.9%	69.9%
CiteSeer	1 ms	100k	100	72.4%	72.0%	68.5%
PubMed	7 ms	100k	1000	82.4%	80.0%	65.6%

4.4.1 COMPARING WITH RANDOM SEARCH

The random search is already very performant in the sense that the highest accuracy are discoverable at certain hardware constraints. For example, with 1 ms latency, 100,000 LUTs/FFs and 100 DSPs, it achieves the best accuracy among the three methods. However, when the constraints are more narrow the distribution of decent sampels are far more sparse. As a result, the best accuracy covered by searching a fixed number of samples is lower than the other two methods.

The search time of random method is around 1x to 2x of the joint search. There are two explanations for that. Firstly, the sampled newtorks are more scattered so their average size is larger. Although the GPU calls are equal, the training time of randomly sampled networks are higher. Another reason is that in order to reach the same number of implementable samples as joint search, much more episodes needs to be inspected so the CPU time adds up to a coniderable level.

4.4.2 COMPARING WITH SEPARATE SEARCH

The separate search consumes highest time with our setting because 1) more sampels are actually trained due to the manual setup; and 2) the architecture found in the first step is larger than average size. It is observed that accuracy is slightly better than random search and in some cases surpass the joint search. However, since the pure architecture search are not aware of the hardware constraints at all, the post-quantization accuracy may degrade severly as decent bit width allocation hardly exists.

4.4.3 IN-DEPTH OBSERVATION

The experimental results concludes that our method exlpores the design space more efficiently than the baselines. It achieves the best accuracy in most hardware cases while runs 1x - 3x faster. The advantage owes to the fact the SW/HW co-search explores the design space in a local region approaching to the constrained area. Figure 3 plots actual hardware statistics of the searched samples projected onto three usage-latency planes. It is shown the Pareto frontier of the joint search method is closet the valid area constrained by the hardware among all the methods.

5 CONCLUSION

Neural architecture search is a promising solution for the advancentment of graph neural network engineering, but it lacks hardware awareness. In this work we propose to an FPGA-based SW/HW co-design framework, named FGNAS, that jointly explores the architectural and hardware spaces. Using reinforcemnt learning, generic hardware model, and mixed precision design, FGNAS performs evidently more efficient than the random search and traditional separate methods. Under different hardware constraints, FGNAS has the best accuracy in majority of the cases with 1x-3x faster running time. Besides, the cause of the advantage is discussed from statistical analysis.

REFERENCES

Liang-Chieh Chen, Maxwell D. Collins, Yukun Zhu, George Papand reou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for Efficient Multi-Scale Architectures for Dense Image Prediction. *arXiv e-prints*, art. arXiv:1809.04184, September 2018.

Y. Gao, H. Yang, Peng Zhang, Chuan Zhou, and Y. Hu. Graphnas: Graph neural architecture search with reinforcement learning. *ArXiv*, abs/1904.09981, 2019.

Tong Geng, Ang Li, Wang Tianqi, Chunshu Wu, Yanfei Li, Antonino Tumeo, and Martin Herbordt. Uwb-gcn: Hardware acceleration of graph-convolution-network through runtime workload rebalancing, 08 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. *arXiv e-prints*, art. arXiv:1704.01212, April 2017.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

Kexin Huang, Cao Xiao, Lucas Glass, Marinka Zitnik, and Jimeng Sun. Skipgnn: Predicting molecular interactions with skip-graph networks, 2020.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. *arXiv e-prints*, art. arXiv:1511.05493, November 2015.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search, 2019.

Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu. High performance graph convolutionai networks with applications in testability analysis. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2019.

Yuzhe Ma, Zhuolun He, Wei Li, Lu Zhang, and Bei Yu. Understanding graphs in eda: From shallow to deep learning. In *Proceedings of the 2020 International Symposium on Physical Design*, ISPD '20, pp. 119–126, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370912. doi: 10.1145/3372780.3378173. URL https://doi.org/10.1145/3372780.3378173.

Mateusz Nurek and Radosław Michalski. Combining Machine Learning and Social Network Analysis to Reveal the Organizational Structures. *arXiv e-prints*, art. arXiv:1906.09576, June 2019.

Stefan Spalević, Petar Veličković, Jovana Kovačević, and Mladen Nikolić. Hierarchical protein function prediction with tail-gnns, 2020.

Qiaoyu Tan, Ninghao Liu, and Xia Hu. Deep representation learning for social network analysis. *Frontiers in Big Data*, 2:2, 2019. ISSN 2624-909X. doi: 10.3389/fdata.2019.00002. URL https://www.frontiersin.org/article/10.3389/fdata.2019.00002.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization. *ArXiv*, abs/1811.08886, 2018.

Yuke Wang, Boyuan Feng, Gushu Li, Shuangchen Li, Lei Deng, Yuan Xie, and Yufei Ding. Gnnadvisor: An efficient runtime system for gnn acceleration on gpus, 2020.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *arXiv e-prints*, art. arXiv:1901.00596, January 2019.

Shen Yan, Biyi Fang, Faen Zhang, Yu Zheng, Xiao Zeng, Hui Xu, and Mi Zhang. Hm-nas: Efficient neural architecture search via hierarchical masking, 2019.

Hanqing Zeng and Viktor Prasanna. Graphact: Accelerating gcn training on cpu-fpga heterogeneous platforms. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '20, pp. 255–265, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370998. doi: 10.1145/3373087.3375312. URL https://doi.org/10.1145/3373087.3375312.

Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pp. 161–170, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450333153. doi: 10.1145/2684746.2689060. URL https://doi.org/10.1145/2684746.2689060.

Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. Fast hardware-aware neural architecture search, 2020.

Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. Auto-GNN: Neural Architecture Search of Graph Neural Networks. *arXiv e-prints*, art. arXiv:1909.03184, September 2019.

Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. *arXiv e-prints*, art. arXiv:1611.01578, November 2016.