Learned Low Precision Graph Neural Networks

Yiren Zhao[*]
yiren.zhao@cl.cam.ac.uk
University of Cambridge

Duo Wang[*]
duo.wang@cl.cam.ac.uk
University of Cambridge

Daniel Bates
daniel.bates@cl.cam.ac.uk
University of Cambridge

Robert Mullins
robert.mullins@cl.cam.ac.uk
University of Cambridge

Mateja Jamnik
mateja.jamnik@cl.cam.ac.uk
University of Cambridge

Pietro Lio
pietro.lio@cl.cam.ac.uk
University of Cambridge

ABSTRACT

Deep Graph Neural Networks (GNNs) show promising performance on a range of graph tasks, yet at present are costly to run and lack many of the optimisations applied to DNNs. We show, for the first time, how to systematically quantise GNNs with minimal or no loss in performance using Network Architecture Search (NAS). We investigate the novel quantisation search space of GNNs. The proposed NAS mechanism, named Low Precision Graph NAS (LPGNAS), constrains both architecture and quantisation choices to be differentiable. LPGNAS learns the optimal architecture coupled with the best quantisation strategy for different components in the GNN automatically using back-propagation in a single search round. On the citation datasets, solving the task of classifying unseen nodes in a graph, LPGNAS generates quantised models with significant reductions in both model and buffer sizes but with similar accuracy to manually designed networks and other NAS results. The reduced latency with quantisation is crucial for the speed of GNN based query answering and the smaller RAM requirements support larger batch sizes and thus a larger service throughput. In particular, on the Pubmed dataset, LPGNAS shows a better size-accuracy Pareto frontier compared to seven other manual and searched baselines, offering a 2.3× reduction in model size and also a 0.4% increase in accuracy when compared to the best NAS competitor.

1 INTRODUCTION

Graph Neural Networks (GNNs) have been successful in fields such as computational biology [33], social networks [6], knowledge graphs [11], *etc.*. The ability of GNNs to apply learned embeddings to unseen nodes or new subgraphs is useful for large scale machine learning systems on graph data, ranging from analysing posts on forums to creating product listings for shopping sites [6]. Most of the large production systems have high throughputs, running millions or billions of GNN inferences per second. This creates an urgent need to minimise both the computation and memory cost of GNN inference.

One common approach to reduce computation and memory overheads of Deep Neural Networks (DNNs) is quantisation [8, 9, 25, 29, 32]. A simpler data format not only reduces the model size but also introduces the chance of using simpler arithmetic operations on existing or emerging hardware platforms [8, 29]. Previous quantisation methods focusing solely on DNNs with image and sequence data will not obviously transfer well to GNNs. First, in CNNs and RNNs, it is the convolutional and fully-connected layers that are quantised [8, 14]. GNNs, however, follow a sample and aggregate approach [6, 30], where we can separate a basic building block in GNNs into four smaller sub-blocks (Linear, Attention, Aggregation and Activation); only a subset of these sub-blocks has trainable parameters. Second, the design of GNN blocks involves a different design space, where several attention and aggregation methods are available and the choices of these methods have a direct interaction with numerical precisions. Third, previous CNN NAS methods [19] consider two possible quantisable components (weights and activations) for a single layer. Given n quantisation options, this provides n^2 combinations. We show that a single GNN layer has n^6 quantisation combinations, offering a significantly larger quantisation search space (Section 3.1).

In this paper, we propose Low Precision Graph NAS (LPGNAS), which aims to automatically design quantised GNNs. The proposed NAS method is single-path, one-shot and gradient-based. Additionally, the quantisation options of LPGNAS are at a micro-architecture level so that different sub-blocks in a graph block can have different quantisation strategies. In this paper, we make the following contributions.

- To our knowledge, this is the first piece of work studying how to systematically quantise GNNs. We identify the quantisable components and possible search space for GNN quantisation.
- We present a single-path, one-shot and gradient-based NAS algorithm (LPGNAS) for automatically finding low precision GNNs.
- We report LPGNAS results, and show how they significantly outperfrom previous state-of-the-art NAS methods and manually designed networks in terms of memory consumption on the same accuracy budget on the citation datasets.

2 RELATED WORK

In this section, we first provide relevant literature in the field of quantisation and network architecture search in Section 2.1. We then review prior work in the field of graph neural networks and explain what recently proposed GNN NAS can achieve in Section 2.2.

2.1 Quantisation and network architecture search

DNNs offer great performance and rapid time-to-market path on a broad variety of tasks. Unfortunately, their high memory and computation requirements can be challenging when deploying in real-world scenarios. Quantisation directly shrinks model sizes and rapidly simplifies the complexity of the arithmetic hardware. Previous research shows that DNN models can be quantised to surprisingly low precisions such as ternary [32] and binary [8].

[*]Both authors contributed equally to this research.

Networks with extremely low precision weights have a significant task accuracy drop due to the numerical errors introduced, and sometimes require architectural changes to compensate [8]. In contrast, fixed-point numbers [9, 14] and other more complex arithmetics [25, 29] have larger bitwidths but offer a better task accuracy. These quantisation methods have primarily been applied on convolutional and fully-connected layers since they are the most compute-intensive building blocks in CNNs and RNNs. Another way of reducing the inference cost of DNNs is architectural engineering. For instance, using depth-wise separable convolutions to replace normal convolutions not only costs fewer parameters but also achieves comparable accuracy [7]. However, architectural engineering is normally tedious and complex; recent research on NAS reduces the amount of manual tuning in the architecture design process. Initial NAS methods used evolutionary algorithms and reinforcement learning to find optimal network architectures, but each iteration of the search fully trains and evaluates many child networks [13, 34], thus needing a huge amount of computation resources and time. Liu *et al.* proposed Differentiable Architecture Search (DARTs) that creates a supernet with all possible operations connected by probabilistic priors [12]. The search cost of DARTs is reduced by orders of magnitude – it is the same as training a single supernet (one-shot). In addition, DARTs is gradient-based, meaning that standard Stochastic Gradient Descent (SGD) now can be used to update these probabilistic priors. One major drawback of DARTs is that all operations of the supernet are active during the search. Recently proposed single-path NAS methods only update a sampled network from the supernet at each search step, and are able to converge quickly and significantly reduce the computation and memory cost compared to DARTs [1, 5, 18]. In addition, many of the NAS methods on vision networks consider mixed quantisation in their search space [5, 19], but limit the quantisation granularity to per-convolution level.

Since GNNs being deployed in the cloud for analysing large-scale knowledge graphs [6], quantisation will be an important method for reducing power consumption and reducing latency when the system is running a large number of GNN inferences on these knowledge graphs per day. The reduced latency with quantisation is crucial for query answering speed and the smaller RAM requirements support larger batch sizes and thus a larger service throughput. While there are currently no well-known GNN applications for embedded devices, the knowledge and understanding gained from a established low-precision Graph NAS algorithm may enable future applications which would not have been possible otherwise.

2.2 Graph neural network

Deep Learning on graphs has emerged into an important field of machine learning in recent years, partially due to the increase in the amount of graph-structured data. Graph Neural Networks has scored success in a range of different tasks on graph data, such as node classification [6, 10, 16], graph classification [20, 23, 28] and link prediction [27]. There are many variants of graph neural networks proposed to tackle graph structured data. In this work we focus on GNNs applied to node classification tasks based on Message-Passing Neural Networks (MPNN) [4]. The objective of the neural network is to learn an embedding function for node features so that they quickly generalise to other unseen nodes. This inductive nature is needed for large-scale production level machine learning systems, since they normally operate on a constantly changing graph with many coming unseen nodes (*e.g.* shopping history on Amazon, posts on Reddit *etc.*) [6]. It is also worth to mention that many of these systems are high-throughput and latency-critical. The reductions in energy and latency of the deployed networks imply the service providers can offer a better user experience while keeping a lower cost. Most of the manually designed GNN architectures proposed for the node classification use MPNN. The list includes, but is not limited to, GCN [10], GAT [16], GraphSage [6], and their variants [2, 15, 17, 26]. These works differ in ways of computing the edge weights, sampling neighbourhood and aggregating neighbour messages. We also relate to works focusing on building deeper Graph Neural Networks with the help of residual connections, such as JKNet [21]. To the best of our knowledge, there is no prior work in investigating quantisation for Graph Neural Networks.

There is a recent surge of interest in looking at how to extend NAS methods from image and sequence data to graphs. Gao *et al.* proposed GraphNAS that is a RL-based NAS applied to graph data [3]. Later, Zhou *et al.* combined the RL-based NAS with parameter sharing [31] and developed AutoGNN. Zhao *et al.* proposed a gradient-based, single-shot NAS called PDNAS, and searched both the micro- and macro-architectures of GNNs [30]. You *et al.*also recently looked at the general design space of GNNs, but the study does not focus on number represetations of GNNs [24]. All of these graph NAS methods show superior performance when compared to other manually designed networks.

3 METHOD

In this section, we first explain the architectural and quantisation search spaces in Section 3.1, and then look at how the LGPNAS algorithm work in Section 3.2.

3.1 Search space

We consider a single graph block, or a GNN layer, as four consecutive sub-blocks (Equation (1)). Equation (1) considers node features h^{k-1} from the $k-1$ layer as inputs, and produces new node features h^k with trainable attention parameters a^k and weights w^k. The four sub-blocks, as illustrated in Equation (1), are the Linear block, Attention block, Aggregation block and Activation block; these sub-blocks have operations as search candidates, which is a similar architectural search space to Zhao *et al.* [30] and Gao *et al.* [3]. We provide a detailed description of the architectural search space in the Appendix. In Equation (2), we label all possible quantisation candidates using Q. The quantisation function Q can be applied not only on learnable parameters (*e.g.*, w^k) but also activation values between consecutive sub-blocks. In addition, we allow quantisation options in Equation (2) to be different. For instance, Q_a can learn to have a different quantisation from Q_w, meaning that a single graph block receives a mixed quantisation for different quantisable components annotated in Equation (2).

$$h^k = \mathrm{Act}(\mathrm{Aggr}(\mathrm{Atten}(a^k, \mathrm{Linear}(w^k, h^{k-1})))) \quad (1)$$

Table 1: Quantisation search space for weights and activations. Frac bits means the number of fractional bits and total bits represent the total bitwidth.

Weights			Activations		
Quantisation	Frac Bits	Total Bits	Quantisation	Frac Bits	Total Bits
binary	0	1	fix2.2	2	4
binary	0	1	fix4.4	4	8
ternary	0	2	fix2.2	2	4
ternary	0	2	fix4.4	4	8
ternary	0	2	fix4.8	4	12
fix1.3	3	4	fix4.4	4	8
fix2.2	2	4	fix4.4	4	8
fix1.5	5	6	fix4.4	4	8
fix3.3	3	6	fix4.4	4	8
fix2.4	4	6	fix4.4	4	8
fix4.4	4	8	fix4.4	4	8
fix4.4	4	8	fix4.8	8	12
fix4.4	4	8	fix8.8	8	16
fix4.8	8	12	fix4.8	8	12
fix4.12	12	16	fix4.4	4	8
fix4.12	12	16	fix4.8	8	12
fix4.12	12	16	fix8.8	8	16

Table 2: Different types of attention mechanisms. W here is parameter vector for attention. $<,>$ is dot product, a_{ij} is attention for message from node j to node i.

Attention Type	Equation
Const	$a_{ij} = 1$
GCN	$a_{ij} = \frac{1}{\sqrt{d_i d_j}}$
GAT	$a_{ij}^{gat} = \text{LeakyReLU}(W_a(h_i \vert\vert h_j))$
Sym-GAT	$a_{ij} = a_{ij}^{gat} + a_{ji}^{gat}$
COS	$a_{ij} =< W_{a1}h_i, W_{a2}h_j >$
Linear	$a_{ij} = \tanh(\sum_{j \in N(i)}(W_a h_j))$
Gene-Linear	$a_{ij} = W_g \tanh(W_{a1}h_i + W_{a2}h_j)$

Table 3: Different types of activation functions.

Activation	Equation
None	$f(x) = x$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
Tanh	$f(x) = tanh(x)$
Softplus	$f(x) = \frac{1}{\beta}\log(1 + e^{\beta x})$
ReLU	$f(x) = Max(0, x)$
LeakyReLU	$f(x) = Max(0, x) + \alpha Min(0, x)$
ReLU6	$f(x) = Min(Max(0, x), 6)$
ELU	$f(x) = Max(0, x) + Min(0, \alpha(e^x - 1))$

$$\begin{aligned} h^k_{\text{linear}} &= Q_l(\text{Linear}(Q_w(w^k), Q_h(h^{k-1}))) \\ h^k_{\text{atten}} &= Q_{at}(\text{Atten}(Q_a(a^k), h^k_{\text{linear}})) \\ h^k &= \text{Act}(Q_{ag}(\text{Aggr}(h^k_{\text{atten}}))) \end{aligned} \tag{2}$$

The reasons for considering input activation values as quantisation candidates are the following. First, GNNs always consider large input graph data, the computation operates on a per-node or per-edge resolution but requires the entire graph or the entire sampled graph as inputs. The amount of intermediate data produced during the computation is huge and causes a large pressure on the amount of on-chip memory available. Second, quantised input activations with quantised parameters together simplify the arithmetic operations. For instance, $\text{Linear}(Q_w(w^k), Q_h(h^{k-1})))$ considers both quantised h^{k-1} and quantised w^k so that the matrix multiplication with these values can operate on a simpler fixed-point arithmetic.

The quantisation search space identified in Equation (2) is different from the search space identified by Wu *et al.* [19] and Guo *et al.* [5]. Most existing NAS methods focusing on quantising CNNs look at quantisation at each convolutional block. In a graph neural network, a single graph block is equivalent to a convolutional block. In a single graph block, we look at more fine-grained quantisation opportunities within the four sub-blocks. The quantisation search considers a wide range of fixed-point quantisations. The weights and activation can pick the numbers of bits in $[1, 2, 4, 6, 8, 12, 16]$ and $[4, 8, 12, 16]$ respectively, and we allow different integer and fractional bits combinations We list the detailed quantisation options in the Table 1. Table 1 shows the quantisation search space, each quantisable operation identified will have these quantisation choices available. BINARY means binary quantisation, and TERNARY means two-bit ternary quantisation. FIX$x.y$ means fixed-point quantisation with x integer bits and y bits for fractions. We also demonstrate the number of fractional bits (FRAC BITS) and total number of bits (TOTAL BITS).

In total, as listed in Table 1, we have 17 quantisation options; and as illustrated in Equation (2), there are six possible components that join the quantisation search, this gives us in total $17^6 = 24137569$ quantisation combinations for a Graph Neural Network.

As illustrated in Figure 1, each graph block consists of four sub-blocks, namely the linear block, attention block, aggregation block and activation block. Each of these sub-blocks contain architectural choices that joins the NAS process. In Table 2, we show all attention types that LPGNAS considers. The attention types includes a various styles of parametric or non-parametric attention methods. In Table 3, we list all activation types that were considered in the architectural search space. Apart from attention and activation types, we also consider searching for the best hidden feature size, using two fully-connected layers with an intermediate layer with an expansion factor that can be picked from a set $\{1, 2, 4, 8\}$. In terms of aggregation, the LPGNAS algorithm also considers choices including *mean*, *add* and *max*.

The considered search space is similar to prior works in this domain [3, 30, 31]. The possible number of architectural combinations of a single layer is $7 \times 8 \times 3 \times 4 = 672$, for an n-layer network, this means there are in total 672^n possible architectural combinations. Combining the quantisation search space mentioned in the previous section, the search space has in total 24137569×672^n options. Assuming the number of layers considered in the search is $n = 4$, this roughly gives us a search space of 10^{17} options.

3.2 Low precision graph network architecture search

We describe the Low Precision Graph Network Architecture Search (LPGNAS) algorithm in Algorithm 1. (x, y), (x_v, x_v) are the training and validation data respectively. M is the total number of

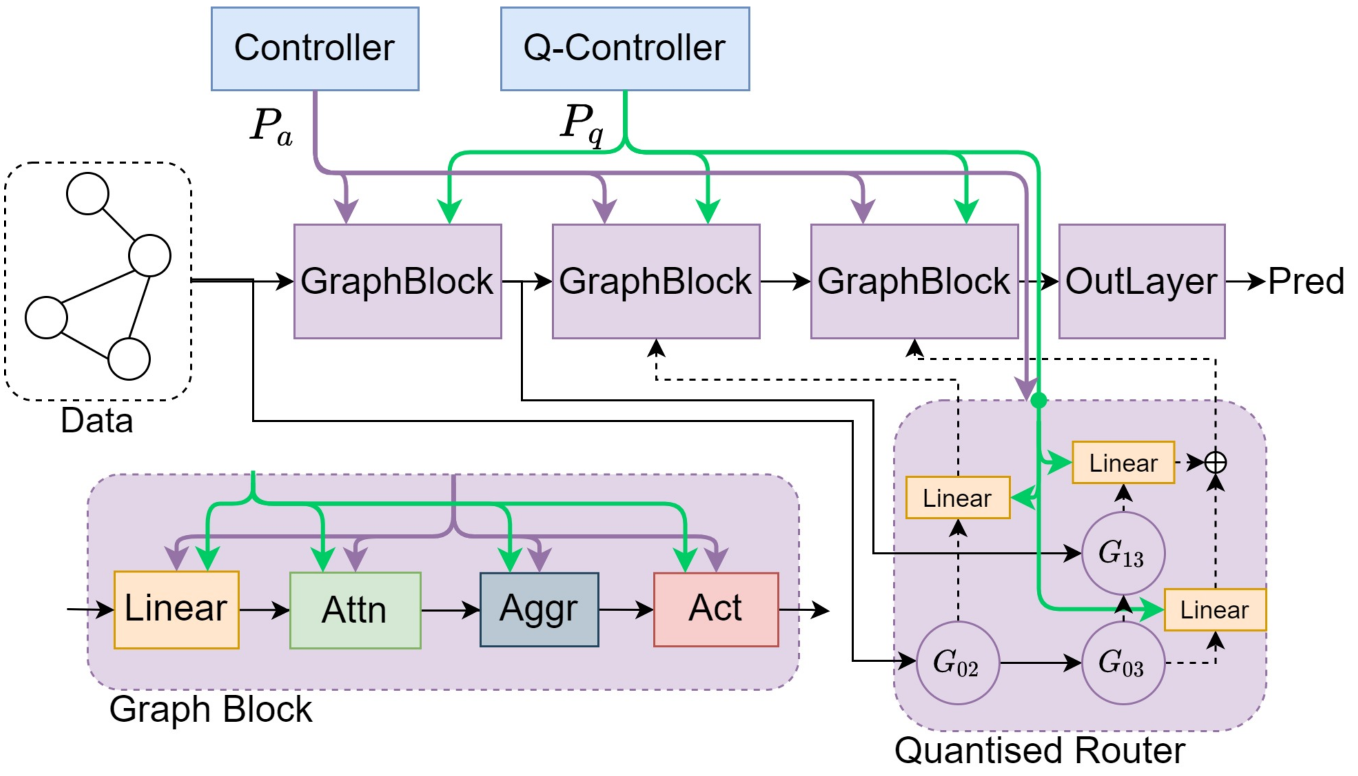


Figure 1: An overview of the LPGNAS architecture. P_a denotes controller output (Purple) for selecting different operations within each graph block, and for routing shortcut connections between graph blocks. P_q denotes quantisation controller (Q-Controller) output (Green) for selecting quantisations for operations within graph blocks and shortcut connections. G_{ij} are gating functions conditioned on P_a. Solid lines are input streams into the router while dashed lines are output streams.

Algorithm 1 LPGNAS algorithm

```
Input: $x, y, x_v, y_v, M, M_a, M_q, K, \alpha, \beta$
Init$(w, w_a, w_q)$
for $i = 0$ to $M - 1$ do
    $N$ = NoiseGen$(i, \alpha)$
    $P_a, P_q = g_a(w_a, x_{\text{val}}, N), g_q(w_q, x_{\text{val}}, N)$
    $\pi_a, \pi_q = \arg\max(P_a), \arg\max(P_q)$
    for $i = 0$ to $K - 1$ do
        $\mathcal{L} = \text{Loss}(x, y, \pi_a, \pi_q)$
        $w = \text{Opt}_w(\mathcal{L})$
    end for
    $\mathcal{L}_v = \text{Loss}(x_v, y_v, \pi_a, \pi_q)$
    if $e > M_a$ then
        $w_a = \text{Opt}_{w_a}(\mathcal{L}_v)$
    end if
    if $e > M_q$ then
        $\mathcal{L}_q = \text{QLoss}(P_a, P_q)$
        $w_q = \text{Opt}_{w_q}(\mathcal{L}_v + \beta L_q)$
    end if
end for
```

search epochs, and M_a and M_q are the epochs to start architectural and quantisation search. Similar to Zhao *et al.* [30], before the number of epochs reaches M_a or M_q, LPGNAS randomly picks up samples and warms up the trainable parameters in the supernet. K is the number of steps to iterate in training the supernet. After generating noise N, we use this noise in architectural controller g_a and quantisation controller g_q together with the validation data x_v to determine the architectural π_a and quantisation π_q choices. After reaching the pre-defined starting epoch (M_a or M_q), LPGNAS starts to optimise the controllers' weights (w_a and w_q). We choose the hyper-parameters $M_q = 20, M_a = 50, \alpha = 1.0, \beta = 0.1$, unless specified otherwise, based on the choices made by Zhao *et al.* [30]. In addition, we provide an ablation study in the Appendix to justify our choices of hyper-parameters. Notice that QLoss estimates the current hardware cost based on both architectural and quantisation probabilities. As shown in Equation (3), we define S as a set of values that represents the costs (number of parameters) of all possible architectural options: for each value s in this set, we multiply it with the probability p_a from architecture probabilities P_a generated by the NAS architecture controller. Similarly, we produce the weighted-sum for quantisation from the set of values S_q that represents costs of all possible quantisations. The dot product $<,>$ between these two weighted-sum vectors produces the quantisation loss L_q.

$$L_q = \text{QLoss}(P_a, P_q) \\ =< \sum_{s \in S, p_a \in P_a} (s * p_a), \sum_{s_q \in S_q, p_q \in P_q} (s_q * p_q) > \quad (3)$$

Figure 1 is an illustration of the LPGNAS algorithm. We use two controllers (g_a and g_q) for architecture and quantisation (named as Controller and Q-Controller in the diagram) respectively. The controllers use trainable embeddings connected to linear layers to produce architecture and quantisation probabilities. The architecture controller provides probabilities (P_a) for picking architectural options of graph blocks and also the router. The router is in charge of determining how each graph block shortcuts to the subsequent graph blocks [30]. In addition, the quantisation controller provides quantisation probabilities (P_q) to both modules in graph blocks and the linear layers in the router. In a graph block, only the linear and attention layers have quantisable parameters and have matrix multiplication operations; neither activation nor aggregation layers have any trainable or quantisable parameters. However, all input activation values of each layer in the graph block are quantised. The amount of intermediate data produced during the computation causes memory pressure on many existing or emerging hardware devices, so even for modules that do not have quantisable parameters, we choose to quantise their input activation values to help reduce the amount of buffer space required.

4 RESULTS

We compare LPGNAS to both manually designed and searched GNNs on the Citation datasets, where the network takes the entire graph as an input in a transductive learning setup [22].

The Citation datasets include nodes representing documents and edges representing citation links. The task is to distinguish which research field the document belongs to [22]. We use both manually designed and searched networks from other NAS frameworks as our baselines. For the manually designed baselines, we exhaustively searched for the best quantisation combinations for the Citation datasets.

The search terminates when quantisation causes a decrease in accuracy bigger than 0.5% and rolls back to pick the previous quantisation. For all reimplemented baselines, we train the networks with three different seeds and report the averaged results with standard deviations. For LPGNAS, we run the entire search and train cycle three times and report the final averaged accuracy trained on the searched network.

4.1 Citation datasets

Table 4 shows the performance of GraphSage [6], GAT [16], JKNet [21], PDNAS [30] and LPGNAS on the citation datasets with a partition of 0.6, 0.2, 0.2 for training, validation and testing respectively. For the quantisation options of GraphSage, GAT and JKNet, we manually perform a grid search on the Cora dataset. The grid search considers various weight and activation quantisations in a decreasing order. We reimplemented GraphSage [6], GAT [16] and JKNet [21] for quantisation.

The results in Table 4 suggest that LPGNAS shows better accuracy on both Cora and Pubmed with quantised networks. In addition, although LPGNAS does not show the smallest sizes on these two datasets, it is only slightly bigger than the manual baselines but shows a much better accuracy. On Citeseer, LPGNAS only shows slightly worse accuracy (around 0.1% less) with a considerably smaller size (around 9× reduction in model sizes).

To further prove the point that LPGNAS generates more efficient networks, we sweep across different configurations and different quantisation regularisations to produce Figure 2 to visulise the performance of LPGNAS and how it performs with small model sizes. Using different base network structure configurations and different architectural and training learning rates, we can control a trade-off between model sizes and accuracy. We then plot the Pareto frontiers of LPGNAS putting model sizes on the horizontal axis and accuracy on the vertical axis for Pubmed (Figure 2) by generating a group of searched models with different model size budgets. Our proposed method shows a Pareto dominance compared to all evaluated methods, meaning that it strikes the best combinations between model sizes and accuracy. In this plot, we also demonstrate how buffer size trades-off with accuracy. For buffer size, it means the total size required to hold all intermediate activation values during the computation of a single inference run with a batch size of one. GNNs normally use large graphs as input data; the amount of memory required to hold all intermediate values might be a limiting factor in batched inference. Since LPGNAS uses quantisation not only on weights but also on input activation values, it shows a better trade-off than the floating-point baselines. For Figure 2, models staying at the top left of the plots are considered as better, since they are consuming less hardware resources and maintaining high accuracy. In addition, in Figure 2, we compare PDNAS [30] to LPGNAS and found that we outperformed the floating-point NAS methods by staying at the top left in the plot.

4.2 Quantisation Search Time

To further illustrate that LPGNAS has significantly reduced the amount of search time required, we provide Table 5 to show how LPGNAS compares to a fixed JKNet-32 architecture in terms of the amount of GPU hours spent for searching for the best quantisation options. Because of the limited resources we have, we estimate the quantisation search cost of a JKNet-32 by running 5 different randomly selected quantisation combinations and multiply the averaged training time with the total quantisation options.

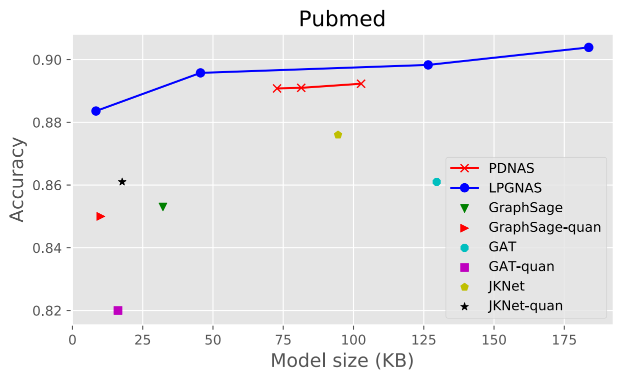
Table 5 shows that LPGNAS can significantly reduce the search time. For instance, on Citeseer, the search time can be reduced by around 2270×. It is worth noting that, when performing this quantisation search time comparison, we do not consider the architectural search space of the baseline, meaning that the baseline (JKNet) is a fixed-architecture. LPGNAS also searched for architecture choices. This further increases the search time of the baseline if this additional architectural space is considered.

5 CONCLUSION

In this paper, we propose a novel single-path, one-shot and gradient-based NAS algorithm named LPGNAS. LPGNAS is able to generate compact quantized networks with state-of-the-art accuracy. To our knowledge, this is the first piece of work that systematically

Table 4: Accuracy and size comparison on Cora, Pubmed and Citeseer with a data split of 60%, 20% and 20% for training, validation and testing. Our results are averaged across 3 independent runs. The numbers in bold show best accuracies or smallest sizes.

Method	Quan	Cora Accuracy	Cora Size	CiteSeer Accuracy	CiteSeer Size	PubMed Accuracy	PubMed Size
GraphSage	float	74.5 ± 0.0%	92.3KB	75.3 ± 0.0%	237.5KB	85.3 ± 0.1%	32.2KB
GraphSage	w10a12	74.3 ± 0.1%	28.8KB	75.1 ± 0.1%	74.2KB	85.0 ± 0.0%	10.1KB
GAT	float	88.9 ± 0.0%	369.5KB	75.9 ± 0.0%	950.3KB	86.1 ± 0.0%	129.6KB
GAT	w4a8	88.8 ± 0.1%	46.2KB	68.0 ± 0.1%	118.8KB	82.0 ± 0.0%	16.2KB
JKNet	float	88.7 ± 0.0%	214.9KB	75.5 ± 0.0%	505.2KB	87.6 ± 0.0%	94.5KB
JKNet	w6a8	88.7 ± 0.1%	40.3KB	73.2 ± 0.1%	94.7KB	86.1 ± 0.1%	17.7KB
PDNAS-2	float	89.3 ± 0.1%	192.2KB	**76.3 ± 0.3%**	478.6KB	89.1 ± 0.2%	72.8KB
PDNAS-3	float	89.3 ± 0.1%	200.0KB	75.5 ± 0.3%	494.4KB	89.1 ± 0.2%	81.4KB
PDNAS-4	float	89.8 ± 0.3%	205.0KB	75.6 ± 0.2%	500.0KB	89.2 ± 0.1%	102.7KB
PDNAS-4	w8a8	86.9 ± 0.1%	51.3KB	69.3 ± 0.1%	125.0KB	88.9 ± 0.1%	25.7KB
PDNAS-4	w12a16	88.8 ± 0.2%	76.9KB	74.4 ± 0.2%	187.5KB	89.0 ± 0.1%	38.5KB
LPGNAS	mixed	**89.8 ± 0.0%**	67.3KB	76.2 ± 0.1%	**56.5KB**	**89.6 ± 0.1%**	45.6KB



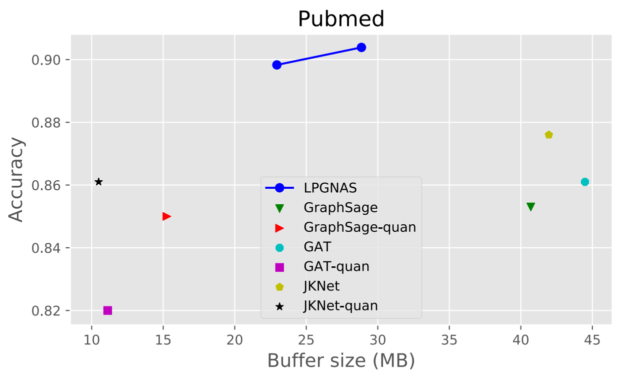


Figure 2: Pareto Frontier of LPGNAS and PDNAS (only floating-point versions) [30] on the Pubmed datset [22], the manually designed networks are shown as dots. On the left, we show the trade-off between model sizes and accuracy; on the right, the trade-off is between buffer sizes and accuracy. Buffer sizes mean the amount of memory space required to store temporary activations.

Table 5: Search cost in GPU hours, all experiments are conducted on an NVIDIA GeForce RTX 2080 Ti GPU.

Dataset	Cora	Citeseer	Pubmed
LPGNAS	3.2	3.6	4.2
JKNet-32	6518.5	8165.6	5289.1

studies the quantisation of GNNs. We define the GNN quantisation search space and show how it can be co-optimised with the original architectural search space. The end results demonstrate that a co-optimisation between the architectural and quantisation spaces greatly improves network accuracy. The searched networks show pareto dominance on a accuracy model size trade-off over all manually designed networks.

REFERENCES

[1] Han Cai, Ligeng Zhu, and Song Han. 2018. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332* (2018).

[2] Jianfei Chen, Jun Zhu, and Le Song. 2017. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568* (2017).

[3] 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).

[4] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1263–1272.

[5] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2019. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420* (2019).

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[7] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets:

Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[8] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in neural information processing systems*. 4107–4115.

[9] Kyuyeon Hwang and Wonyong Sung. 2014. Fixed-point feedforward deep neural network design using weights +1, 0, and −1. In *Signal Processing Systems*.

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

[11] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-ninth AAAI conference on artificial intelligence*.

[12] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1eYHoC5FX

[13] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2902–2911.

[14] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. Q-bert: Hessian based ultra low precision quantization of bert. *arXiv preprint arXiv:1909.05840* (2019).

[15] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. 2019. Two-stream adaptive graph convolutional networks for skeleton-based action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 12026–12035.

[16] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations* (2018). https://openreview.net/forum?id=rJXMpikCZ

[17] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *The World Wide Web Conference*. 2022–2032.

[18] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. FBNET: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 10734–10742.

[19] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. 2018. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090* (2018).

[20] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*. https://openreview.net/forum?id=ryGs6iA5Km

[21] 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*. 5449–5458.

[22] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*. JMLR. org, 40–48.

[23] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*. 4800–4810.

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

[25] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. 2018. LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks. In *European Conference on Computer Vision (ECCV)*.

[26] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. 2018. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294* (2018).

[27] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*. 5165–5175.

[28] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[29] Yiren Zhao, Xitong Gao, Daniel Bates, Robert Mullins, and Cheng-Zhong Xu. 2019. Focused Quantization for Sparse CNNs. In *Advances in Neural Information Processing Systems*. 5585–5594.

[30] Yiren Zhao, Duo Wang, Xitong Gao, Robert Mullins, Pietro Lio, and Mateja Jamnik. 2020. Probabilistic Dual Network Architecture Search on Graphs. *arXiv preprint arXiv:2003.09676* (2020).

[31] 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).

[32] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2017. Trained ternary quantization. *International Conference on Learning Representations (ICLR)* (2017).

[33] Marinka Zitnik and Jure Leskovec. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* 33, 14 (2017), i190–i198.

[34] Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. (2017).