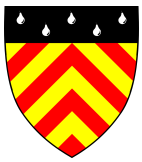
UNIVERSITY OF CAMBRIDGE

Accelerating Neural Networks for Graph Structured Data

PhD Proposal

Shyam A. Tailor



Clare Hall

First year report submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Contents

Chapter 1

Introduction

The success of deep learning in a wide variety of application domains has led to research addressing efficiency concerns and systems challenges of handling these workloads. Innovation exists on multiple levels: specialised accelerators exclusively for neural networks, efficient neural network architectures, and other hardware-software co-design techniques to improve model efficiency. This research has lead to the opportunity to deploy neural network models pervasively for applications such as computer vision or natural language processing.

Recently, there has been a trend in building machine learning methods that operate on non-uniform structured data, such as graphs. Although this field is in its infancy relative to other better established fields, we can already see early applications of graph neural networks (GNNs) [167] in industry. For example, DeepMind and Google recently announced their success in applying GNNs to travel time prediction in Google Maps, achieving a 50% improvement over the previous technique used [2]. In the literature, these architectures have been shown to achieve excellent results in fields ranging from computer vision [37, 180, 141, 129] and natural language processing (NLP) [73, 113, 188, 187, 112, 15] to areas such as drug design [29] or code analysis [5, 6, 45, 151]. Performance is rapidly improving on these tasks, as newer GNN architectures are developed, and new applications for these models are published regularly. Additionally, there are firm links between GNNs and the Transformer model [155] which has become dominant in NLP applications: a Transformer encoder and decoder may be interpreted as special cases of GNNs acting on graphs with regular topologies. Transformers can be viewed as the first runaway success story for GNNs, but many more will follow in future years.

Given the growing success of GNNs, it is natural to ask whether we can develop the techniques that enables these algorithms to run with lower latency, consume less memory and energy, and run on a wider variety of hardware. However, there are many challenges

that must be considered. Unlike other common architectures, we have to deal with *non-uniform* inputs. This represents a significant challenge, since we are unable to make certain optimizations up-front without knowledge of the structure of the input data. We must design techniques that can gracefully handle a wide variety of topologies. Another issue with GNNs is *sparsity*. Most GNN models involve propagating information based on the structure of the input graph—but in many cases the adjacency matrix is mostly empty: density can often be less than 0.1%. Optimising for irregular sparse operations is more difficult than optimising for regular dense operations that appear in other types of neural network. Another challenge is the size of the intermediate activations produced by GNNs, which is a function of the size of the input graph. These can consume large amounts of memory—limiting which devices GNNs can be deployed to—and cause increases in energy consumption which is dominated by data movement costs. Finally, there is often substantial redundant computation being performed with naive implementations of GNNs, which increases energy consumption and latency.

This proposal focuses on hardware-software co-design techniques to improve the efficiency of GNN architectures. These techniques are have underpinned the revolution enabling deep learning models to be deployed to mobile devices in the NLP and computer vision domains. However, applying existing techniques to GNNs directly is difficult, and GNNs provide new, previously unexplored, challenges. For example, one common optimisation is quantization: using low-precision integer arithmetic at inference time. This technique can significantly reduce memory usage and data movement costs, and often reduces inference latency. Despite strong advances in quantization techniques for CNNs and Transformers, naively applying these techniques to a GNN yields poor results [149] as nodes in the graph may not have the same degree, causing unequal distributions for activations. There are also many techniques that are relevant to GNNs that are less important for these other architectures. For example, activation compression would be extremely beneficial for reducing GNN inference time on CPUs by improving the cache hit rate, and could significantly reduce data movement. These techniques are of secondary interest for other common architectures. This proposal covers these techniques, and others to improve efficiency.

During my first year at Oxford, I developed an interest in GNNs after discovering first hand the difficulties associated with training them. I published two papers last year: the first argues that modern computer vision methods should be applied on wearable devices for activity recognition; GNNs are known to be beneficial to classification performance in this application, however they are not yet viable on these types of devices. The second paper makes a first step towards improving the viability of these architectures by showing how to apply quantization without a dramatic loss in accuracy—even down to 4 bits.

This document starts by covering related work to this proposal, before proceeding to cover work that was completed in my first year. Finally, the proposed work for the upcoming years of the PhD is described, along with a timeline.

Chapter 2

Related Work and Background

This section begins by analysing systems for executing traditional graph algorithms, including coverage of common optimisations that can be applied. GNNs are covered next, including theory, outstanding problems, applications, and implementation concerns. Hardware-software co-design techniques, such as quantization, pruning and architectural design principles that have become popular for CNNs and Transformers are covered next. Finally, neural accelerators, including graph-focused designs, are covered.

2.1 Systems for Traditional Graph Analysis

A graph $\mathcal{G} = (V, E)$ is a structure consisting of a set of vertices, V, which are related to each other in some way by edges, E. Graphs represent a fundamental data structure in Computer Science, and there has been decades of study devoted to them. The reader is referred to an introductory algorithms text for further background if required [41]. The rest of this section will focus on systems that have been developed for executing algorithms on large-scale graphs with billions, if not trillions, of edges. Developments in these systems will be covered in roughly chronological order since they arose a decade ago.

Efficient graph processing is difficult: graph algorithms exhibit poor locality in memory accesses, and many algorithms perform relatively little work per node. As a result, memory accesses are the bottleneck when executing these algorithms. Additionally, division of work is difficult, as graphs are non-uniform.

Most systems expose a *vertex-centric scatter-gather* programming model. In this model, the programmer must "think like a vertex" and send messages to other vertices, and handle incoming messages to update to a new state. This model is sufficiently expressive that many algorithms of practical interest can be encoded into it.

Pregel [114] was one of the earliest systems for large scale distributed graph processing, developed by Google to continue scaling PageRank [127] after the limitations of MapReduce's [46] functional paradigm were becoming clear. Pregel operates with the bulk-synchronous parallel model, with the vertex programs running in parallel in each superstep, followed by communication between vertices. Computation ceases when all vertices have voted to stop the computation. This is a simple computation model admitting less complicated implementation, but it does prevent certain optimisations being applied. GraphLab [110] is another early system which employs a shared-memory model. Through analysis of the data dependency graph resulting from the vertex program, the system's scheduler executes the algorithm out-of-order, with the formal guarantee that the result will be identical to the sequential execution. PowerGraph [65] scaled GraphLab to distributed clusters by considering how to minimise the overhead from between-node communication. Although at first glance the obvious solution is to minimise the number of cut edges, this is practically difficult: algorithms for finding these cuts are relatively slow (comparable to program runtime), and perform badly on power-law graphs[1]. Instead, it was better to consider vertex cuts: *evenly assign edges to machines, and allow vertex duplication*. At the end of a program step, the vertex duplicates are reconciled.

GraphChi [97] and X-Stream [140] contrast from these works by assessing how far it is possible to scale graph computation while limited to a single machine with a large quantity of persistent storage. GraphChi recognised the importance of avoiding random accesses to edges, which would incur a slow random disk access. Instead, it is better to take advantage of sequential streaming bandwidth. The system employs the concept of shards: subsets of the graph consisting of vertices with all their incoming and outgoing edges; a pre-processing step sorts the shard edges by their source vertex to enable better usage of sequential bandwidth. X-Stream takes this idea further, by changing the programming model to be *edge-centric*, enabling even greater utilisation of streaming bandwidth, and without requiring a costly sorting step.

Cagra [181] is another single-machine graph system, focusing more closely on the gap between cache and DRAM performance—rather than memory and disk. This work introduces CSR segmenting, where vertices are divided to fit into the last-level cache of the CPU. This approach, unlike other approaches described, produces intermediate results that must be merged afterwards; this may require program re-designs.

Jia et al. [82] propose Lux, a distributed multi-GPU system for fast graph processing. As GPUs offer far-superior memory bandwidth (even for random accesses) they are well suited to accelerating graph workloads. There are several new challenges to consider, however: firstly, GPUs have a significantly different memory hierarchy than CPUs, meaning

[1] Referring to the node degree distribution

strategies for data placement and transfer do not translate well. Secondly, the push-based vertex scatter model that other systems use inhibits runtime optimisations that GPUs use. Lux rejects the vertex partitioning scheme used by PowerGraph, and instead uses a simple edge-partitioning scheme, where the edge-list is evenly partitioned by vertex ID, with the partitions being moved over time to allow for work balancing. Lux also moves to a pull-based programming model, rather than a push-based programming model. These strategies enables the GPU's memory access coalescing hardware to be used optimally, boosting runtime performance.

System	Target	Novelty
Pregel	Distributed	Bulk synchronous parallel, vertex-centric model running on a fleet of machines.
GraphLab	Single Machine	Shared memory system with scheduling mechanisms to enable the specification of programs running on multiple threads without requiring complicated locking protocols.
PowerGraph	Distributed	Extension of GraphLab to distributed setting by minimising communication cost through the use of vertex-cuts.
GraphChi	Single Machine	Maximise usage of sequential bandwidth from disk by dividing graphs into shards which can be processed in a self-contained way; apply a sorting step to the shards to further optimise sequential bandwidth.
X-Stream	Single Machine	Introduce edge-centric programming model, to enable sequential passes through the graph edges without expensive pre-processing steps.
Cagra	Single Machine	Division of the graph into sub-graphs that fit into last-level cache. Parallel, cache-aware merging of partial results produced by sub-graphs.
Lux	Distributed GPU	Vertex-cut partioning like PowerGraph, with dynamic reallocation based on load. Pull-based vertex-centric programming model to take advantage of GPU hardware.

Table 2.1: Summary of a variety of different graph-processing systems.

This section has covered existing systems for large-scale graph analysis, and identified core design principles underpinning them; this is summarised in table 2.1. It is worth noting that single-machine systems do not have significantly different performance to their distributed brethren. As mentioned earlier, graph processing tends not to be compute-bound, but rather memory or communication bound. As McSherry et al. [116] identify, there are substantial costs to moving to a distributed system, and we should not automatically assume that a distributed system will outperform a well-tuned non-distributed system.

2.1.1 Common Optimisation Techniques

This section will cover other commonly used graph optimisations and design principles that are applied in the context of graph analysis.

Graph Reordering [8, 17, 88, 98, 99, 163, 10]. Reordering the input graph is useful to improve memory locality. The cost of this may be amortised over several runs, although it is common for overhead of reordering to be hundreds of times larger than the speedup achieved over the baseline [10]—often dramatically reducing the utility of these methods.

(Cache) Blocking [181, 68, 147]. Rather than processing the whole graph at once, it is possible to break it up into multiple subgraphs, and perform a merge phase. However, this may require rewriting the program, unlike graph reordering. Cagra [181] is one example of this approach.

Vertex Scheduling. Rather than reordering the graph, it is also possible to reorder the vertex processing order. It has been shown that using schedules generated by Hilbert Space Filling Curves [72] can significantly improve cache performance with little overhead [116, 181]. These schedules improve locality at both source and destination vertices, rather than just one like sorting the edges would achieve.

2.2 Graph Neural Networks (GNNs)

Traditional graph processing methods have relied on hand-crafted algorithms and features [66]. Recently, however, the rise of DL-based methods for graphs allows us to learn representations that are sensitive to the task and input graph, which can offer significant boosts to performance as observed in the CV and NLP domains. This section will cover theory, outstanding problems, applications and implementation challenges. We also discuss the links between Transformers and GNNs.

Graph Embeddings. Prior to the emergence of GNNs, there existed techniques for embedding graphs into a vector, which could be consumed by downstream applications. A graph embedding algorithm can take graphs with, or without, edge and node features, and produces embeddings for nodes, edges, or the entire graph. Early methods usually involved factorization of a similarity matrix (e.g. the Laplacian or adjacency matrix) by minimising $\mathcal{L}(\mathbf{Z}) = ||\mathbf{Z}^\top\mathbf{Z} - \mathbf{S}||_2^2$ [69]. Examples of these techniques include the graph factorization algorithm [4], Laplacian eigenmaps [12], GraRep [26], and HOPE [126].

The rest of this section will cover GNNs, which can be viewed as an evolution of these earlier graph embedding techniques which incorporate ideas from deep learning.

2.2.1 Theory

Many popular GNN architectures may be viewed as generalizations of CNN architectures to an irregular domain: at a high level, graph architectures attempt to build representations

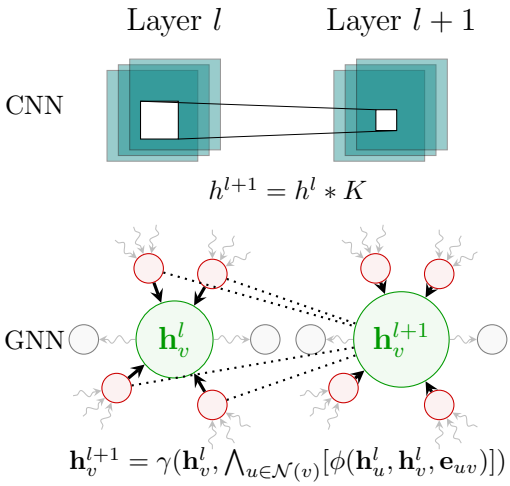


Figure 2.1: Graph convolutions, compared to spatial convolutions used in CNNs.

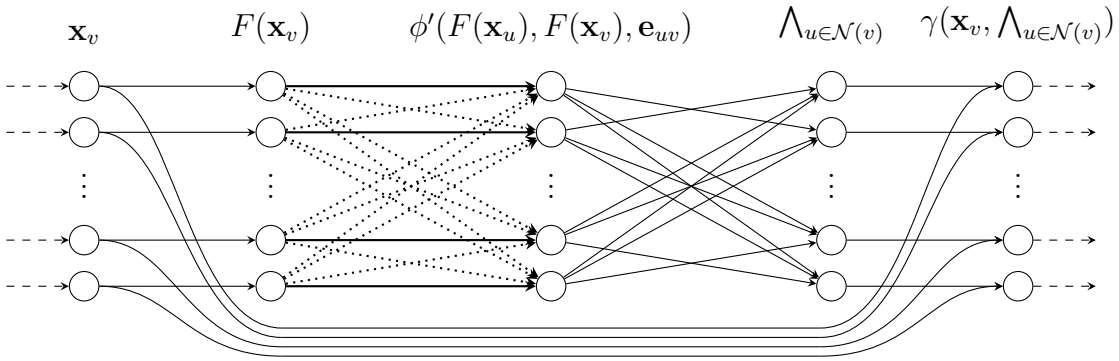


Figure 2.2: Dataflow in the message passing framework for GNNs.

based on a node's neighbourhood. Unlike a CNN, however, this neighbourhood does not have a fixed ordering or size. This is shown visually in fig. 2.1.

It is common to cast GNNs in the message passing framework [64]. A graph $\mathcal{G}$ has node features $\mathbf{X} \in \mathbb{R}^{N \times F}$, an incidence matrix $\mathbf{I} \in \mathbb{N}^{2 \times E}$, and optionally D-dimensional edge features $\mathbf{E} \in \mathbb{R}^{E \times D}$. The forward pass through a GNN layer consists of message passing, aggregation and update phases: $\mathbf{x}'_v = \gamma(\mathbf{x}_v, \bigwedge_{u \in \mathcal{N}(v)}[\phi(\mathbf{x}_u, \mathbf{x}_v, \mathbf{e}_{uv})])$. Messages from node u to node v are calculated using function ϕ, and are aggregated using a permutation-invariant function $\bigwedge$. The features at v are subsequently updated using γ. This procedure is shown visually in fig. 2.2. Several of these layers can be stacked to increase model capacity, and to increase the horizon over which a vertex will receive messages from. The final node representations can then be used for downstream tasks such as node, edge or graph regression or classification.

2.2.1.1 Early Architectures

The earliest works in the GNN literature appeared around 2009, and functioned by repeatedly applying the same set of parameters over the graph (i.e. they are recurrent) [142, 58, 104, 44]. Of particular note are Gated Graph Neural Networks [104], which remain state-of-the-art for applications involving representations of code, and which have an update rule of:

$$\mathbf{h}_v^{(t)} = \text{GRU}(\mathbf{h}_v^{(t-1)}, \sum_{u \in \mathcal{N}} \mathbf{W}_e \mathbf{h}_u^{(t-1)}) \tag{2.1}$$

where $\mathbf{W}_e$ represents the weight matrix corresponding to the edge type between u, v.

2.2.1.2 Spectral Methods

Rather than repeatedly applying the same weights, convolutional-style GNNs consist of multiple layers, each with their own set of weights. Convolutional GNNs fall into two categories: *spectral* methods, which are motivated by graph signal processing (GSP) [144], and *spatial* methods which are motivated by information propagation through the graph.

A fundamental matrix in GSP is the Laplacian: $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the diagonal matrix of node degrees, and $\mathbf{A}$ is the adjacency matrix. It is common to apply symmetric normalisation to this matrix: $\widetilde{\mathbf{L}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$. Since $\widetilde{\mathbf{L}}$ is positive semidefinite, it is possible to diagonalise this matrix as $\widetilde{\mathbf{L}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$. A graph signal $\mathbf{x} \in \mathbb{R}^{|V|}$ is a vector with a single value per node. The graph Fourier transform is then defined as:

$$\hat{\mathbf{x}} = \mathcal{F}(\mathbf{x}) = \mathbf{U}^\top \mathbf{x} \tag{2.2}$$

With the corresponding inverse transform defined as:

$$\mathbf{x} = \mathcal{F}^{-1}(\hat{\mathbf{x}}) = \mathbf{U}\hat{\mathbf{x}} \tag{2.3}$$

We can then define the graph convolution operator $*_\mathcal{G}$ by analogy to the familiar convolution theorem in digital signal processing:

$$\mathbf{x} *_{\mathcal{G}} \mathbf{g} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{g})) \tag{2.4}$$
$$= \mathbf{U}(\mathbf{U}^\top \mathbf{x} \odot \mathbf{U}^\top \mathbf{g}) \tag{2.5}$$
$$= \mathbf{U}\mathbf{g}_\theta \mathbf{U}^\top \mathbf{x} \tag{2.6}$$

As in CNNs, we must learn the filters $\mathbf{g}_\theta$. ChebNet [47] approximates this filter by using Chebyshev polynomials: $\mathbf{g}_\theta \approx \sum_{k=0}^{K_{\max}} \theta_k T_k(\widetilde{\mathbf{\Lambda}})$, where $\widetilde{\mathbf{\Lambda}}$ is the matrix of eigenvalues normalised to $[-1, 1]$. The reader should note that the choice of $K_{\max}$ dictates how many hops a node can aggregate information from. The widely cited Graph Convolutional Network (GCN) [91] is ChebNet with $K_{\max}$ set to 1, enabling for an efficient implementation as $\mathbf{H} = \mathbf{SX\Theta}$, but requiring multiple layers to be stacked to allow information to propagate over multiple hops.

There have been several extensions to GCN. One notable example is Simple Graph Convolution (SGC) [166], which is predicated on the hypothesis that the mixing of information by the convolution—and not the non-linearities between layers—is what underpins the expressiveness of GCN. Under this hypothesis, a stack of GCN layers becomes $\mathbf{S} \dots \mathbf{SX\Theta_1} \dots \mathbf{\Theta_k} = \mathbf{S}^k\mathbf{X\Theta}$. In practice, this approximation is reasonable, with minimal loss in performance, while being significantly faster for training and inference.

2.2.1.3 Spatial Methods

More recently, there has been a shift towards spatial based methods, which are not necessarily motivated by graph signal processing. These methods tend to consider a wider variety of aggregation techniques at a node, are more theoretically sound for directed graphs, and in practice obtain better results [49].

Message Passing Neural Networks (MPNNs) [64] were motivated as a technique to describe these spatial-based GNNs. It is worth noting, however, that we are able to cast GCN in this framework too. Noteworthy architectures include:

1. **GraphSAGE** [70], which provided several innovations, including treating the central node at each step differently to its in-neighbours, the introduction of aggregation functions, and the concept of sampling neighbouring nodes to reduce memory consumption.

2. **Graph Attention Network** (GAT) [156], which popularises the concept of anisotropy—that each neighbouring vertex should not necessarily be weighted equally, through the use of an attention mechanism.

3. **Graph Isomorphism Network** (GIN) [168], which is motivated by the fact that many predecessor works used aggregation functions that are susceptible to creating identical embeddings for non-isomorphic graphs. The authors propose using a learnable parameter ε to weight the central node, and prove that this modification increases the expressive power of the architecture.

4. **Principal Neighbourhood Aggregation Network** (PNANet) [42], which proposes the use of multiple aggregation functions and weighting methods in a single network. Each aggregation method can cover for other aggregations' weaknesses, hence reducing the risk of confusing non-isomorphic neighbourhoods; this technique resulted in a large performance boost on several benchmarks.

One recent survey paper [49] benchmarked many popular architectures on a variety of standardised datasets. In practice, the best performing architectures were *anisotropic*, with the absolute best architecture (GatedGCN [19]) adding gating—where the weightings at the previous layer are used in the computation of the next layer's weightings. PNANet, with a wide range of aggregation functions, achieves similar performance.

2.2.1.4 Outstanding Problems

One major issue in the GNN literature is the difficulty of training deep GNNs. It is well known that training GNNs of depth beyond even 2-4 layers causes a dramatic drop in performance. As with other architectures, there is the issue of vanishing gradients and overfitting. However, there is an additional cause: *oversmoothing*. As recognised by Li et al. [102], GCNs can be shown to be a special case of Laplacian smoothing (i.e. neighbourhood averaging). It can be shown that stacking many layers will lead to the oversmoothing phenomenon, where all nodes in a connected component collapse to the same values; this dramatically reduces downstream performance. Although proofs generally focus on GCN, it has been empirically shown that most architectures suffer from this issue [28], with GAT being one of the worst affected. This issue remains an open problem, with several techniques being proposed [184, 139, 30, 190]. In general, however, these methods do not show that going deeper improves performance substantially.

Another challenge is to introduce directionality into the architecture. This topic has been explored in Beaini et al. [11], and could be particularly relevant for applications such as point clouds.

Predictions over edges, and handling edge-based features also remains an issue. One common approach to handling edge features is to concatenate the edge features to the source vertex's features during message generation. Other approaches include modifying the aggregation function to incorporate edge features [77] or using different weight matrices

to generate messages based on edge type [104].

2.2.1.5 Implementation Challenges

Efficient implementation of GNNs involves several different challenges to those presented by other common architectures such as CNNs or language models.

Non-Static Inputs. With a CNN the input dimensions are known upfront: in the case of images, the image dimensions and colour channels are known upfront. This can enable optimisations to be made up front from the dataflow graph, where the shapes[2] of each node are known. One example optimisation is device placement optimisation: choosing how to partition the computation graph across several accelerators [191, 121]. In contrast, input graphs do not have this level of regularity; device placement optimisation, for example, would not be a reasonable optimisation to apply.

Scatter-Gather versus Sparse Matrix Multiplication. The message passing paradigm that has been widely adopted fits closely with scatter-gather implementation, where the messages are *explicitly materialised* and scattered to their destination node. While intuitively appealing, this approach causes significant memory overhead, and may not be necessary. GCN, for example, can be implemented with (sparse) matrix multiplication alone—dramatically reducing memory usage, and often resulting in faster training and inference. In contrast, GAT does require these messages to be explicitly materialised due to the attention mechanism requiring features derived from both the source and destination node to be concatenated to create the attention coefficients. From an efficiency perspective, the clear preference is towards avoiding architectures which require this explicit materialisation; however, given that it is known that anisotropy is a major contributor to performance, which—thus far—requires the materialisation, it may be the case that future high-performance GNN architectures will require efficient support for this mechanism.

Model Size and Memory Consumption. As we can see in fig. 2.4, the model sizes for various GNN models is substantially smaller than common CNN models. Unlike CNN models, however, the memory (and operations) used during inference and training depends on the input size; whether the memory usage scales with vertices or edges is dependent on the choice of GNN architecture. In practice, memory consumption is a major bottleneck towards scaling GNNs further; common accelerators (e.g. GPUs) typically have a relatively low quantity of memory, with the quantity often used to perform market segmentation. Another practical consideration is that modern mobile SoCs now include neural accelerators, but which are further constrained memory-wise. This bottleneck is most commonly hit

[2] often ignoring batch size

when attempting to train with a large single graph; in the literature, the largest dataset that is commonly assessed is Reddit [70], however this graph is relatively small compared to many of the publicly available datasets (e.g. the Amazon product graph [85]). In addition to techniques such as mixed-precision training, or using low-precision arithmetic at inference time, there are graph-specific techniques to reduce memory consumption. One example is to randomly sample nodes at each epoch of training; this was introduced by GraphSAGE [70], although newer techniques exist [177]. However, as shown by Jia et al. [85] in their attempt to build a distributed GNN training platform, these methods do cause a noticeable reduction in model accuracy. It remains an open question whether we can train on such large graphs without necessarily needing to use multiple accelerators, or performance inference using neural accelerators which cannot fit the entire graph into memory at once.

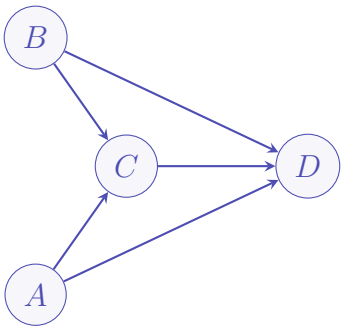


Figure 2.3: Duplicated computation in GNNs. The computation at node C is a subset of that done at D.

Redundant Computation. During the forward pass of a GNN, there may be substantial duplicated work that is performed. To give an explicit example, consider the graph in fig. 2.3. If we use a summation aggregation function, then we can see that the work performed at node C is a subset of the work that is performed at node D, and the result at D could be computed from the result at C. This redundancy was exploited by [84], which demonstrated that hierarchically restructuring the graph upfront can boost inference and training performance by 2×. This work is an interesting first attempt, but has significant upfront cost[3] which is non-trivial to parallelise, and uses a hardware-unaware cost function to decide splits. In practice, while this technique may be reasonable for training, it is unlikely to be of any real benefit for inference on unseen graphs. A similar approach is also implemented in the Rubik accelerator [33], but this also requires a high pre-processing cost which is not reported.

Edge Features. As mentioned in the previous section, it is currently unclear how to handle edge features. However, it appears that existing approaches to handling edge

[3]sufficiently high that the authors do not report it

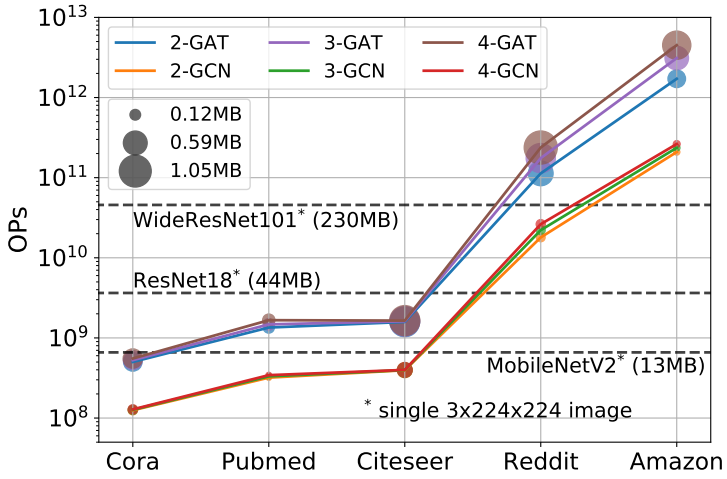


Figure 2.4: Model size versus number of operations for GAT and GCN on a variety of input graphs.

features requires the scatter-gather paradigm, rather than sparse matrix multiplication, similar to the issue described earlier in this section regarding attention coefficients, due to the need to explicitly materialise the message. Compared to anisotropic networks, there will be poorer input re-use for edge-centric networks (reducing the impact of cache performance) since there will be little re-use across different messages; in vertex-centric approaches, messages sent between from each node are identical: anisotropic architectures simply apply a different weighting to each message. This will affect strategies for acceleration, since prioritising re-use is a weaker consideration.

2.2.2 Applications

Due to their versatility in modelling non-uniform data, there has been an explosion in applications these architectures have been used for. This section will present just a brief selection, chosen for their need for support for large-scale training or fast inference.

Computer Vision. GNNs are commonly paired with the output of CNNs to aid in applications where some form of higher scene understanding is required. Examples from CVPR 2020 include feature matching across two images of the same scene for use in estimating a homography [141], activity recognition from videos [180, 37], or video captioning [129].

Natural Language Processing. GNNs have been applied to tasks such as morphology [73] and named entity recognition [113]. In addition, they are state of the art on fake news detection and automated fact verification [188, 187, 112, 15].

Programs and Compilers. Allamanis et al. [5] demonstrated that GNNs could be used for code-related purposes. In their work, they focused on tasks such as detecting

variable misuse: where a variable is used, and typechecks, but is actually not what the developer intended. Other works include learning a type checker for Python [6], tooling for reverse engineering of binaries [45], and learning how to fix build errors [151]. Mendis et al. [118] demonstrated that GNNs could be applied to find better superword-level parallelism heuristics in the LLVM [100] backend. Unlike other application areas, gated graph neural networks [104], which are recurrent—not convolutional—are the most popular architecture choice.

Recommender Systems and Social Network Analysis. Ying et al. [173] described a recommender system developed by Pinterest, which they observed to offer higher performance (150% improvement) than other methods evaluated, despite significant approximations being necessary due to computational limitations. More recent work has assessed how to fuse information across multiple graphs [53]. Snapchat have also described a system for modelling user engagement [108]. Other work has described how to predict the influence a user has in a social network [136], and whether a content item will become popular within a social network [25].

Point Clouds. Point clouds commonly arise in the analysis of 3D data; for example, they are the output of LIDAR sensors. Many state of the art techniques for point cloud analysis involve constructing graphs from point clouds, and running GNNs over the graph. GNN-based approaches achieve state of the art performance on point cloud classification and segmentation [169], along with object detection in point clouds [143].

2.2.3 Transformers as Graph Neural Networks

Observation	Graph Neural Networks	Transformers
Oversmoothing	Training GNNs beyond depth 3-4 is difficult [102] due to representations converging to the same value	Representations in transformers become smooth and individual token identifiability declines [22]; training deep transformers is difficult [107]
Information Mixing	Wu et al. [166] show empirically that the effectiveness of GCNs is due to information mixing rather than non-linearities	Mandava et al. [115] find that attention layers are more valuable early in the architecture
Value of Attention	Anisotropic architectures can be matched by non-anisotropic architectures [42]	There are many proposals for the replacement of self-attention [165, 105, 152, 174], and numerous optimizations applied to attention (expanded in text).

Table 2.2: Comparison of observations made for GNNs and Transformers. There are clear trends across the two architectures.

Transformers [155] can be viewed as a type of graph neural network, specialised for

sequence processing. As explained in the introduction, they may be viewed as the great success story for GNNs, albeit in disguise. GPT-3 [21] has demonstrated that scaling these models enables hitherto impossible applications; however, as noted by Thompson et al. [153], the scaling costs imply that it will be difficult, if not impossible, to train future iterations of GPT models. Surprisingly, there remains relatively little understanding of how transformers are functioning in practice, and several unexpected results have been obtained. For example, while the excellent performance is commonly attributed to the attention mechanism, there have been several recent works which empirically demonstrate that self-attention can be *completely* replaced [165, 105, 174, 152], but the same is not true of cross-attention. It is worth asking whether any approaches pioneered in the GNN literature can be transferred across to Transformers, in order to improve efficiency through architectural modifications, *and vice-versa*.

Although it is commonly claimed that attention coefficients can be used as a method for model interpretation, several studies have shown that attention coefficients should be studied with care, as they do not correspond closely with other methods for interpreting models [80, 164, 22]. In particular, it is clear that manipulating attention coefficients does not tend to result in loss in downstream accuracy [80].

Compete removal of these attention coefficients can often be beneficial. Wu et al. [165] showed that it is possible to replace self-attention with a convolutional operation. Lioutas and Guo [105] demonstated that using box filters, with widths predicted per timestep, instead of self-attention was also sufficient to beat a transformer. Tay et al. [152] achieved similar results to the baseline by learning an attention matrix as a parameter, rather than computing it. Finally, You et al. [174] showed that it was possible to replace self-attention with fixed Gaussian kernels with little degradation in performance. However, none of these works were able to replace cross-attention, in addition to the self-attention. One key observation that the reader should note is that many of these approaches correspond to low-pass filtering information in the time domain. There is evidence suggesting that GNNs also tend to perform low pass filtering [125].

Other works have shown that it is possible to sparsify attention matrices substantially with little penalty [38, 92, 13, 176]. Another line of research is low-rank approximation of the attention matrix, or using kernel methods [162, 90, 39]. Both these lines of research demonstrate the redundancy of attention. It has also been shown that relatively few attention heads contribute to overall model accuracy; it is possible to prune most heads with minimal degradation in accuracy [119]. As noted by Corso et al. [42], a single method of aggregating information at a node from its neighbours is sub-optimal: adding further aggregation methods adds to the model's ability to discriminate between two neighbourhoods. Multihead attention can be interpreted as a method of adding multiple

aggregators, however given that it is known that many heads can be pruned without penalty, it is reasonable to suggest that it is not an effective method for increasing discriminative power at a node. Given that we are aware that attention mechanisms alone are not enough to achieve state of the art performance for GNNs [49], it is reasonable to ask whether any innovations from the GNN literature can be applied to Transformers.

A major obstacle to accelerating Transformers is also the wide fully connected (FC) layers [79], due to their large memory bandwidth requirements. In practice, it is possible to significantly reduce the impact of these layers on inference latency by swapping to grouped convolutions [79][4]. It remains unclear, however, why it is necessary for such wide FC layers to be used, and why it is difficult to stack several narrower layers instead. It is known that tokens in the sequence gradually become smoother as model depth increases [22], as in GNNs. Liu et al. [107] attempt to train deep transformers by choosing an initialization which stabilises gradients, and achieve noticeable improvements to BLEU score [130], albeit at the cost of an increase of nearly an order of magnitude in parameters. DeLighT [117] also attempts to train deeper transformers, although the real contribution of this work is in learning that parameters are more efficiently spent in the later layers of the encoder and decoder.

Early mixing of information appears to be critical to performance in Transformers. One work introduced the idea of sandwich transformers [135], where there are greater numbers of self-attention layers at low depth, and conversely, greater numbers of FC layers at high depth. Mandava et al. [115] evaluate this concept more systematically through neural architecture search, observing the same effect and further concluding that FC layers should outnumber attention layers with a ratio of 5 to 1. These observations regarding information mixing are reminiscent of those made by SGC [166], where they show that GCN's apparent performance is due to information mixing, rather than non-linearities.

2.3 Co-Design of Hardware and Algorithms

To get the best performance—latency, energy, memory usage—from the hardware, it is necessary to design algorithms that suit the characteristics of the hardware. In this section various techniques for improving algorithmic efficiency are discussed.

2.3.1 Low-Bit Representations

It is common to use 32-bit floating point for training and inference of neural networks. In recent years, there has been interest in using fewer bits for these use-cases. The benefits

[4]The reader should note that a position-wise FC layer can be re-interpreted as a pointwise convolution.

of using fewer bits are:

1. Reduced data movement, which is the biggest driver of energy consumption [74].
2. Reduced storage requirements for weights (*model compression*), and reduced runtime memory consumption of activations.
3. Favourable characteristics for accelerator design: simpler, more energy efficient and faster processing elements, lower memory requirements.
4. Reduced latency in situations where the accelerator is memory-bound.

16-bit floating point formats are common and widely supported on the latest accelerators for training [87, 120]. There is also research on moving to 8 bits for training [161], although at time of writing these approaches do not appear practical.

For inference, use of 8-bit integer (fixed point) arithmetic has become commonplace. There have been several studies into varying bit-widths [93, 109]; it is clear that 8 bits results in minimal loss in accuracy and is suitable for existing hardware. The de facto approach to training accurate quantized models is to use quantization-aware training (QAT); this involves modelling the effect of quantization in the forward pass. Since this rounding operation is non-differentiable, the gradient is approximated using the straight-through estimator (STE) [14], which passes the gradient through unmodified. It is most common to use affine quantization, which involves 2 parameters: $x_{\min}, x_{\max}$, which are used to derive scale factor Δ, and zero point z depending on the number of bits. Quantized values are calculated as: $\text{clamp}(0, 2^{\text{bits}}, \text{round}(x/\Delta) + z)$. Other approaches include product quantization, where columns of matrices are decomposed into vectors which are coded with a codebook [145]. There is interest in increasing accuracy at 4 bits, and the extreme case of 1 bit (binary neural networks) [43]. One recent approach to improve accuracy at 4 bits involved stochastic masking of the weights on the forward pass: masked weights were left unquantized [52]. In general, however, 4 bit arithmetic still introduces substantial degradation to accuracy.

The quantization literature tends to focus on Transformer and CNN models. GNNs are not typically evaluated, although some early works have explored the area [159, 54, 185]. These works have several flaws:

1. They never evaluate whether their technique generalises to unseen graphs.
2. The baselines chosen are not sufficiently strong, as accuracy is sensitive to choice of quantization implementation—unlike CNNs or Transformers [149].
3. The quantization schemes found are difficult to accelerate.

The work I did in my first year [149] addresses all these challenges, and evaluates over a much broader set of architectures and datasets than previous works.

2.3.2 Sparsity

Using sparsity as a method to reduce memory consumption and latency is an emerging topic in the deep learning literature. It is highly relevant to graphs, which usually have sparse adjacency matrices.

2.3.2.1 Pruning

Pruning is the procedure of removing parameters from an existing network, with the typical goal of simultaneously minimising the effect on pruned model's accuracy and minimising the model size [16]. Pruning can be used for model acceleration, but it is not always the case that pruned models can be executed more quickly than other types of models. The technique first arose in the late 1980s [81, 122, 123, 89], although there has been a resurgence of interest in the past 5 years. We can categorise pruning methods using the following features:

1. **Structure**. Unstructured pruning removes individual weights, which is not conducive to acceleration. Structured pruning removes entire neurons or channels.
2. **Scoring**. How the weights are scored for pruning e.g. by magnitude.
3. **Scheduling**. What proportion of the weights are pruned at each step.
4. **Fine-Tuning**. Whether or not the method uses a fine-tuning stage.

The lottery ticket hypothesis states that a dense network contains subnetworks that, when trained in isolation, reach the same accuracy as the original dense network. These networks tend to be around 10-20% ($\sim 8\times$) of the size of the original network; above this percentage, the subnetworks tend to achieve higher accuracy than the original dense network. It is important to comment upon the sparsity levels achieved by pruning, relative to those which are commonly observed with GNNs: in deep learning, it is common to refer to 80-90% sparsity as "sparse", whereas with graph adjacency matrices the sparsity may be closer to 99.9-99.99%. These two sparsity levels require radically different approaches to implementation; in practice, for sparsities of 80-90%, there is often no benefit to switching to sparsity optimised operators on general purpose hardware.

It is observed that sparse models tend to outperform dense ones at a given parameter count [16, 51], although this does not outweigh the benefits of moving to a more parameter-efficient architecture [16]. There has been some progress in training models that are sparse

from initialisation time [101, 158, 150, 51], however these methods do not find optimal subnetworks [55], and are unstructured—hence difficult to accelerate and not offering a reduction in memory consumption. It has been noted, however, that wider sparse networks achieve better accuracy than the dense network with a similar parameter count [51].

Structured pruning is highly relevant in the context of GNNs, as it reduces memory consumption—reducing data movement and latency on some architectures.

2.3.2.2 Acceleration of Sparse Models

There has been work on accelerating the models obtained from unstructured pruning methods [34, 50, 57, 103, 132]. There are two major trends that are worth identifying:

1. **CPUs Benefit More From (Unstructured) Sparsity Than GPUs**. Sparse operations involve many random memory accesses; GPUs do not optimise for this kind of use-pattern, as they attempt to hide slow memory access through frequent thread switches rather than through caching mechanisms. Recent work [57] has indicated that carefully tuned CUDA kernels can achieve equivalent performance at sparsity levels greater than 71%, however their approach was tuned exclusively for a *single GPU model*, and does not generalise easily.
2. **Unstructured Methods Cannot Achieve the Theoretical Gains Implied by FLOPS Alone**. In Elsen et al. [50], which focuses on (mobile) CPUs, we see a real-world speedup that is typically less than half the speedup implied by FLOPs alone.

In general, despite claims from the pruning literature that the technique can be used for achieving latency reductions, the gains are relatively small in many contexts, as the research focus remains primarily on unstructured techniques. NVIDIA's recently introduced A100 GPU [1] is one of the first devices to leverage sparsity for acceleration purposes, but it can only accelerate sparse matrices with a 2:4 pattern, where only 2 elements out of every four in a vector are non-zero, which is a relatively low compression level.

2.3.2.3 Sparsifying Activations

Although most research attention has focused on weight matrices, there has also been work on sparsifying activation maps. This problem is highly relevant to GNNs, which have significant runtime memory consumption. Due to the widespread use of the ReLU activation function [124], activation maps tend to be at least 50% sparse; Rhu et al. [138] measured average sparsity to be 63% across a variety of CNNs. Sparsity was found to vary significantly among architectures, and to increase deeper in the network. In principle, sparser activations

should be easier to compress, resulting in lower runtime memory consumption. As with weights, sparser activations can yield latency reductions [63, 96].

Georgiadis [63] proposed applying an L1 loss to activation maps to induce sparsity (60% over the baseline), and demonstrates that these sparser maps can be compressed. Kurtz et al. [96] builds on this work by using Hoyer regularisation [76], and increasing the ReLU threshold for further sparsity on a per-layer basis. Neither method induces *structured* sparsity: to the best of the author's knowledge, no scheme exists to encourage hardware-friendly sparsity. This topic represents a significant opportunity for further latency reduction on architectures such as CNNs or language models.

2.3.3 Architectural Considerations

Careful architectural design can substantially reduce computational requirements with little impact on accuracy. To date, most research has focused on CNNs, although more recent work has also looked at Transformers.

Common techniques for reducing computational overhead for convolutions include reducing the filter size [78]; reducing the number of input channels (by pointwise-convolution insertion) [71, 78]; and reducing the number of output channels. Grouped convolutions are another technique involving the division of the convolution into several groups, which do not interact with each other [94]; this approach was popularised by MobileNet [75], which used *depthwise-separable* convolutions for which the number of groups equals the number of input channels. To enable information to mix across channels, MobileNet uses pointwise convolutions, although channel shuffling has also been shown to be effective [179].

As previously described, there is a myriad of research into how to make Transformer models more efficient. It has also been shown that the expensive (wide) pointwise dense layers can be decomposed using the grouping approach used in CNNs [79]. Due to the similarities between Transformers and GNNs, there may be opportunities to borrow ideas from each architecture and apply it to the other.

To the best of the author's knowledge, there has been no work on designing architectural elements for graph neural networks which consider metrics other than accuracy, such as memory or latency.

2.3.3.1 Neural Architecture Search

Neural Architecture Search (NAS) [192] can be used to automatically find good architectures. Early works focused exclusively on accuracy, but the emergence of *hardware-aware* NAS allows for co-optimisation of metrics such as latency and memory consumption,

alongside accuracy.

Early NAS techniques employed reinforcement learning (RL) to search over the search space [192], with later works adopting techniques such as weight sharing [134] or hypernetworks [20] to reduce the overhead of sampling. More recent work has focused on differentiable NAS [106], which dramatically reduces the overhead of sampling since all models in the space are effectively trained at once. These methods have substantial memory consumption, hence they must search over a proxy task. ProxylessNAS [23] reduces the memory cost, albeit with a significant runtime penalty as the search space grows, and adds latency as a metric to differentiable NAS. More recent work has proposed techniques to accelerate ProxylessNAS while retaining its memory footprint [157]. Other recent work has proposed how to train models suited for deployment across a variety of devices, while training just one model [24]. Dudziak et al. [48] demonstrated that GNNs could be applied to (gradient-free) NAS for latency prediction and accuracy prediction (via a learned binary relation on accuracy).

NAS has been applied to find GNN architectures. Gao et al. [59] proposed an RL-based NAS for GNNs; a similar approach using parameter sharing was proposed in Zhou et al. [189]. Both these approaches considered microarchitecture: that is, the architecture inside a layer, and not the connections between layers. Zhao et al. [186] propose a differentiable NAS scheme that considers both these issues. None of these works demonstrate how to train networks which generalise to unseen graphs, and instead focus on semi-supervised learning. There has also been no work on hardware-aware NAS for GNNs.

Closely related to NAS is the concept of discovering quantization policies for architectures. HAQ [160] is an RL-based technique for discovering a flexible quantization policy, subject to model size and latency constraints. Zhao et al. [185] assessed a differentiable approach for discovering quantization policy in GNNs, but, as with their earlier work, constrained their work to semi-supervised learning.

2.4 Accelerator Design

DNN accelerators have become popular in both academia and real-world deployments. Designing specialist hardware for applications can yield significant benefits in energy consumption and latency, although it is worth noting that there limits to the achievable speedups without optimisation at the algorithmic level [56]. To date, most accelerators have focused on acceleration of linear layers and convolutions [1, 86, 32, 178, 133, 131, 35, 36]. In general, there are two major properties that a DNN accelerator aims to fulfill:

1. **Reducing Data Movement**. Data movement accounts for the majority of energy

Operation	Energy / pJ
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mul	0.2
32b Mul	3.1
16b FP Mul	1.1
32b FP Mul	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

Table 2.3: Relative energy consumption for common operations used in accelerators [74]. Note that memory accesses are orders of magnitude more expensive than compute.

use, as shown by table 2.3. Reducing the number of times data is fetched (e.g. through reuse, maximised by tiling), and how expensive each fetch is (e.g. through low precision) both contribute to reducing energy consumption.

2. **Saturating Processing Elements**. With DNNs, memory bandwidth is usually the bottleneck.

The concept of *dataflow* describes how the accelerator processes the elements. In DNN processing, there is substantial flexibility to choose the ordering. Three common approaches are used:

1. **Weight Stationary**. Maximise reuse of weights. Examples include NVDLA [1] and Google's TPU [86].

2. **Output Stationary**. Maximise reuse of partial sums. Relatively few works use this approach [32, 178, 133].

3. **Input Stationary**. Maximise reuse of input activations. SCNN, which is designed to accelerate sparse networks uses this approach [131].

Less common is the *row stationary* dataflow, which aims to co-optimise for all three datatypes; this approach is used by Eyeriss [35, 36]. However, this approach introduces further complexity into the compiler and hardware to implement.

2.4.1 Sparse Matrix Multiplication and Graph Acceleration

Accelerators for sparse matrix multiplication (SpMM)—a common primitive for GNNs—have been proposed by academia. It is common for works to conflate SpMM and GNNs,

which can be misleading: these works are typically SpMM accelerators, which are evaluated on GNN datasets.

OuterSPACE [128] is a SpMM accelerator which uses the uncommon outer-product approach to matrix multiplication. As recognised by Zhang et al. [182], this leads to significant data movement, as partial matrices need to be stored back to main memory; this work proposes several optimisations to an outer-product based design, most notably the pipelining of the multiply and merge stages where possible, and use of a Huffman tree to minimise data movement. Geng et al. [62] proposes an accelerator which can handle row-level imbalance in the sparse matrix at runtime, and hence improve overall throughput through better load balancing.

Works evaluating both node and graph-level operations include Yan et al. [171], Chen et al. [33] and Auten et al. [9]. Of these works, Chen et al. [33] is the most complete: it clearly delineates between node and graph-level operations, and applies optimisations such as graph reordering and intermediate computation reuse to maximise performance. However, the latency benefits are relatively small: the proposed accelerator is rarely more than 5× faster than the baseline GPU.

None of these works focus on *inference-only* acceleration on smaller, unseen graphs. This is a major oversight in the literature: many optimisations proposed only work after several iterations (e.g. Geng et al. [62]), or require some expensive pre-processing stage which is amortised (e.g. Chen et al. [33]). It is also worth noting that these accelerators primarily optimise for latency, rather than energy or area—which may be a bigger constraint at the edge. Another issue is that these GNN accelerators are SpMM accelerators, and hence do not support more expressive GNN architectures such as GAT.

2.4.2 Processing In- or Near-Memory

It is well known that the energy cost of accessing memory dominates that of computation (table 2.3); by moving the computation closer to memory we can reduce this cost substantially. Near-memory approaches through die stacking are already common, as HBM is used in commercial accelerators [1], although GNNs represent an opportunity to push the compute further into memory. As GNNs tend to have very small weights (typically less than 65KB per layer uncompressed), it may be reasonable to distribute compute amongst memory, rather than using a monolithic design. Approaches like this have shown promise in industry, with UPMEM commercialising DRAM with embedded processors [3].

Chapter 3

Completed Work

In my first year at Oxford I had two papers accepted for publication. Both are included as appendices. The first [154] was on the viability of using imaging sensors for activity recognition in wearable devices, given the strong progress in pushing CNN models down to microcontrollers and progress in designing low power cameras; I am second author on this work, published at HotMobile. The second paper [149] is most related for the work proposed in this report, and describes how to train GNNs which can use low precision arithmetic at inference time. I am joint-first author on this paper, which has been published at the ICML Graph Representation Learning and Beyond Workshop. A full version of this work was rejected at NeurIPS and is currently under review at ICLR.

3.1 Computer Vision on Wearable Devices

"Are Accelerometers for Activity Recognition a Dead-end" [154] argues that now is the time to start investigating whether to replace or augment the accelerometer sensors commonly used in wearable devices with imaging sensors because of recent advances in low-power camera technology, and miniaturising CNNs for computer vision to fit onto low-power microcontrollers. We argue that such wearables are already plausible with existing hardware—and will benefit from future advances in low power accelerators (e.g. the ARM Ethos-U55 [111], which will offer 2 orders of magnitude improvement over current microncontroller hardware). We expect that this approach will enable larger datasets to be collected and labelled, along with providing opportunities for pre-training on common computer vision datasets. We also believe that image data is a richer modality, and would enable a wider variety of activities to be accurately distinguished.

This work did not discuss GNNs, but they are a natural extension. Activity recognition is one area of computer vision that GNNs have been applied to [180, 37]—albeit, not usually

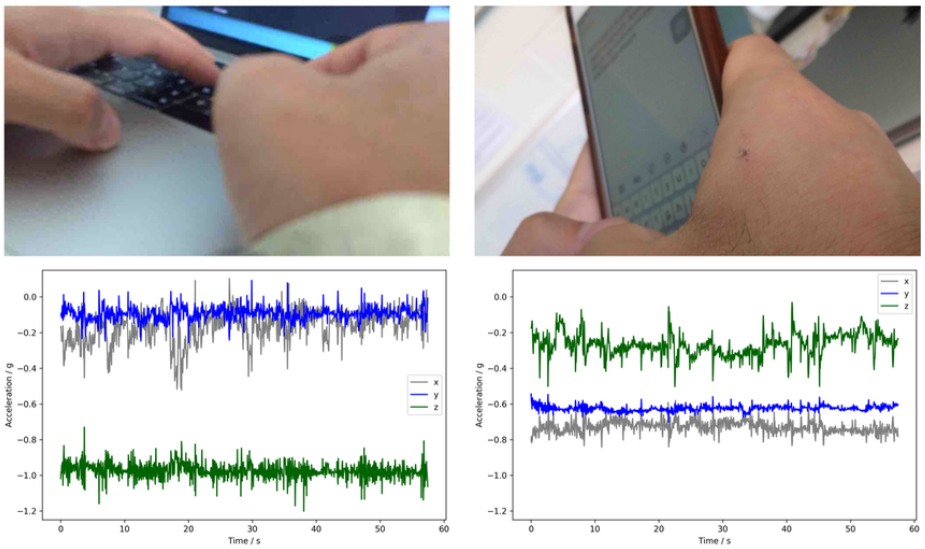


Figure 3.1: Image and accelerometer data collected from the wrist for two different activities. We observe that the accelerometer data is more difficult to classify than the corresponding images. Reproduced from Tong et al. [154].

from a first-person perspective. Another useful application would be pose estimation from images [141].

3.2 Quantization-Aware Training of Graph Neural Networks

"Degree-Quant: Quantization Aware Training for Graph Neural Networks" [149] provides three major contributions:

1. An analysis explaining why existing quantization techniques used for CNNs and language models do not achieve strong results on GNNs.

2. The proposal of a quantization technique that enables the training of GNN models with little or no degradation when using 8-bit integer arithmetic for both weights and activations. At 4-bit arithmetic, our method obtains 26% improvements over the baselines.

3. Initial work on demonstrating the speedups achievable using low-precision with GNNs. We find that with 8-bit arithmetic we observe up to 4.7× reductions in latency on CPUs, with the expected 4× reduction in runtime memory usage.

The core empirical observation made in this work in summarised in fig. 3.2. In summary, we observe that quantization range statistics become distorted, and inaccurate weight updates occur, due to the impact of nodes with high in-degrees. There are several avenues of future

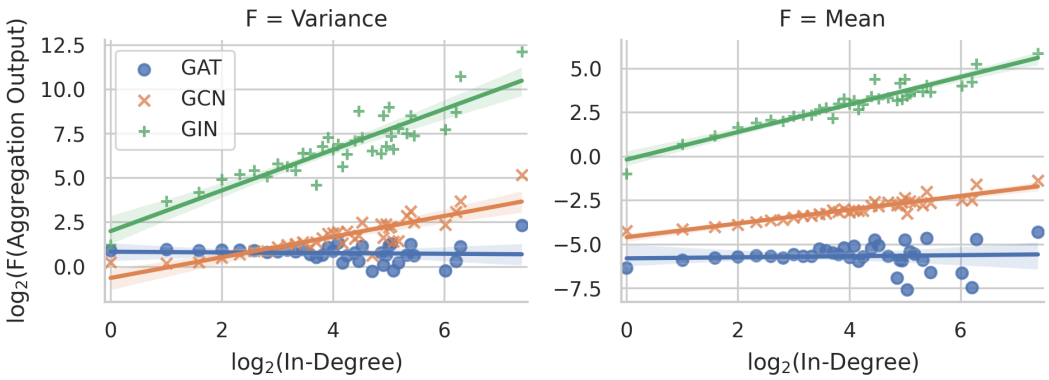


Figure 3.2: GNNs suffer from large variance in values immediately post-aggregation, depending on the choice of aggregation function, as not all nodes have the same in-degree. This makes the challenge of quantizing GNNs much more difficult than CNNs, where the effective in-degree is fixed. The high in-degree nodes contribute most to weight-update errors, and cause the range statistics used for affine quantization to become distorted, causing an overall reduction in accuracy.

work arising from this study: these will be elaborated in the following chapter.

3.3 Other Areas Studied

In addition to the two published works, I also evaluated other projects. These have not yet yielded strong results; I will include an analysis of why these projects failed, along with future work.

3.3.1 Code Optimisation using Graph Neural Networks

As mentioned in the background, GNNs have been successfully applied to modelling code [5]. It is also known that reinforcement learning can be applied for system optimisation [121]. The initial research question in this project was whether it was possible to model the intermediate representation in a compiler using a graph, and learn to optimise it using reinforcement learning (RL). The first pass at this project was to learn a policy to predict which optimisation passes should be applied, and in which order; this problem is known as phase ordering [95]. There had been initial work showing that RL-based approaches to phase ordering were sensible [67].

It was observed that a GNN could rapidly learn a policy that could predict valid optimisation passes to apply to the code. However, it was not possible to learn a policy that could optimise for runtime memory usage or execution time. This is likely due to the reward signal being too noisy for the RL algorithm to learn a useful policy. Another challenge observed was the difficulty of training GNNs over large graphs; this issue was

what prompted my research into accelerating GNN training and inference.

With the benefit of hindsight, there are several ways I would change my approach:

1. Specialise the problem more carefully. One sensible objective would be to find tiling factors for implementations of algorithms such as matrix multiplication or convolution. This is highly relevant to current ML workloads, and could yield insights that could transfer to other types of problem. TVM [31] provides much of the framework to test this problem. This approach would provide more stable memory and latency information.

2. Reject RL in favour of gradient-based optimisation. Although RL has shown some promise for some problems, it has low sample efficiency and has not been shown to converge better than many other search techniques in high-dimensional spaces, such as Bayesian optimisation on problems such as neural architecture search [172]. Another approach would be to assess the viability of using *differentiable surrogates*, and gradient-based optimisation. It is becoming clear that we can build accurate latency prediction functions using neural networks [48, 137], which are differentiable, with relatively few samples. If the latency prediction model is fixed after training, then the problem becomes one of finding the optimal inputs to the model; these could be found using gradient-based optimisation—potentially including second-order optimisation, which can converge in few iterations.

3.3.2 Attention as Low-Pass Filtering

In an ongoing collaboration I am exploring how Transformers behave empirically. The original hypothesis was that the multi-head attention mechanism is low-passing the input tokens in many applications. This hypothesis cannot be true in general, since it is known that Transformers are universal function approximators [175]. However, under assumptions that the attention matrix is low rank—empirically justified by Wang et al. [162]—and that the attention matrix is banded (meaning attention is local)—justified by Sukhbaatar et al. [146]—we can show theoretically that this low-pass behaviour does occur in the transformer encoder's self-attention. However, this behaviour does not occur for the self-attention mechanism in the decoder, which is causal. Introducing causality to the existing assumptions results in high-frequency components being retained.

As predicted by the theory, we found it was possible to simplify the encoder in a transformer; using a PNANet-inspired architecture [42], which is known to achieve high performance on GNN tasks and is efficient to implement, we observed that we could obtain an increase of 0.5 in BLEU score on the IWSLT-14 German-to-English dataset [27] over a baseline transformer with more parameters, a notable rise but not sufficient to be publication

worthy. We found that applying the same technique in the decoder significantly harmed performance, as would be expected from the theory.

We are currently investigating whether our observations can be applied to CNNs. Empirically it is known that these architectures are sensitive to texture, rather than shape i.e. they are sensitive to high frequencies rather than low frequencies [61]. Although it has been shown that self-attention can be used to completely replace convolutional layers [7, 40, 183], doing so is extremely expensive. At time of writing, there is little research exploring the interplay between these two operation, which appear empirically to have different frequency-characteristics; we hope that combining the two operations can boost model efficiency, especially at low parameter-counts. We are evaluating this hypothesis with NAS on CIFAR-10.

3.4 Other Work and Achievements

My Part III project, completed under Cecilia Mascolo, was awarded best paper at the WellComp workshop at UbiComp [148] in September 2020. I have also served as a reviewer for IMWUT/UbiComp.

I was the class tutor for the Computer Architecture course in Oxford between January and March 2020. I am currently supervising Programming in C at Downing and Fitzwilliam colleges in Cambridge.

I also invited Miltos Allamanis from Microsoft Research to Oxford to give the departmental seminar in January.

Chapter 4

Proposed Research

As the background chapter describes, there are several compelling applications for GNNs, ranging from computer vision to code analysis. At present, however, deployments are limited. Training GNNs on large graphs remains difficult, and applying GNNs at inference time is slow. The goal of this PhD is to address several of the system-design issues arising with GNNs, with the end-goal of enabling these applications to be realised in practice through appropriate hardware-software co-design.

The following areas are all worthy of investigation, although it is not expected that all will be covered during the course of the PhD. The first three areas are the primary focus areas; if time allows, however, the extra areas described may also be worth exploring.

1. **Quantization in GNNs**. There are several immediate extensions to assess following my initial work on GNN quantization. Quantization is important since it improves the effectiveness of caches on CPUs, and reduces data movement, which is the dominating factor for energy consumption. One extension would be to investigate whether it is possible to avoid the expensive percentile operation used by the scheme already proposed, using techniques from extreme value theory [18]. The precise impact of topology on quantization of GNNs is also unclear: a larger empirical study would be useful. A related project would be to use GNNs to find quantization policies that transfer, similar to HAQ by Wang et al. [160], but focusing upon transferable policies and fast one-shot optimisation through the use of differentiable surrogates.

2. **Efficient GNN Architecture Design**. To date, GNN architecture designs consider accuracy as the primary objective. However, designing architectures which trade accuracy for substantial improvements in efficiency are also worth investigating; these ideas have been widely explored in the NLP and computer vision domains. One area to assess is whether it is possible to design anisotropic GNNs which do not require

each edge to be materialised explicitly: i.e. causing $\mathcal{O}(E)$ memory consumption. Another worthwhile avenue to explore is whether it is possible to compress a GNN's depth, using techniques such as knowledge distillation, or using modifications to the receptive field; an initial variant of this idea was explored in Yan et al. [170], but was only applied to semi-supervised scenarios. Compressing depth is non-trivial, as with current GNNs, each node has a receptive field that is related to the depth of the architecture.

3. **Activation Compression and Graph Optimisations**. Cache performance is critical for reducing latency when performing inference with GNNs on CPUs. Quantization is one method for increasing the cache hit rate, as more activations can fit in the cache. Compressing these activations will yield further improvements to cache hit rate, along with reducing data movement costs which dominate energy consumption. It would also be prudent to evaluate other optimisations, such as graph reordering or vertex scheduling. Lightweight optimisations which provide improvements for one-shot inference—and are not amortised over many runs—have not been sufficiently explored. These optimisations are likely to impact latency directly, which is a major factor in enabling new applications.

The areas listed above are the primary focus areas during the PhD. The core ideas previously mentioned are mostly focusing on hardware-software co-design techniques for GNNs to enable low-power, low runtime memory usage, and fast inference. However, there are additional areas which are worth investigating which are closely related. These include how to scale training to larger graph sizes on commonly available hardware, how to design energy-efficient hardware for GNN processing, and exploring the applications unlocked by appropriate advances in efficient inference for these architectures.

1. **Large Scale Training**. Training GNNs on graphs which are too large to fit into accelerator memory remains challenging. For other models it is feasible to use approaches such as model or data parallelism [83] to distribute across many accelerators; this approach is not possible with graphs, however, unless the graph contains several disconnect components which do not require synchronisation. Although it is possible to use sampling approaches to reduce memory usage, it has been shown that these approximations significantly reduce accuracy [85]. As highlighted by McSherry et al. [116] it is not safe to assume a distributed solution will exceed the performance of a highly-tuned single-machine implementation: it may be possible to achieve strong performance with careful spilling of data to host memory to reduce accelerator memory pressure. Conversely, there has been little work investigating sparse matrix multiplication, an essential primitive for GNNs, across several accelerators.

2. **Energy-Constrained Acceleration**. Existing work on GNN acceleration primarily focuses on latency. However, this is not the only constraint at the edge: energy is also a primary consideration. Data movement is likely to be the dominating source of energy consumption with GNNs due to the size of the activations. Hardware support for the techniques proposed earlier in the PhD, such as quantization or compression, may help with these movement costs. Additionally, moving compute near-memory could dramatically decrease data movement costs.

3. **Applications**. It would also be interesting to assess real world applications that my work enables; applications in the context of wearable devices would relate closely to my work at HotMobile [154]. Potential applications include activity recognition [37, 180], hand pose estimation [60], or relative pose estimation between two scenes [141].

Chapter 5

Timeline

5.1 Year 2

Michaelmas Term

- Continue work on quantization by addressing immediate follow-ups from Degree-Quant work.
- Implement fast quantized GNN kernels for CPU and potentially other platforms (e.g. mobile or GPU).
- Extensive literature review on activation compression and sparsity.

Lent Term

- Write up conclusions on follow ups to Degree-Quant; if possible, written up early for a conference submission.
- Clean up quantization code and open source as a library.
- Begin work on activation compression. This work may be applicable to other types of architectures, so should evaluate on both GNNs and CNNs or Transformers.

Easter Term

- Write up conclusions from activation compression project.
- Update library and open source.
- Explore follow-ups to activation compression project or assess other types of graph optimisations.

Summer Term

- Buffer time for over-runs from previous projects, or if compelling follow-ups are worth exploring.
- Begin literature review on architecture design.
- Implement ideas to improve GNN architectural efficiency and write up for submission if promising results.

5.2 Year 3

Michaelmas Term

- Finalise literature review for thesis.
- Continue work on GNN architectural efficiency if needed. Otherwise, assess follow-ups for any of previous work completed or choose an extension project.
- Keep library up to date.

Lent Term

- Write up work from previous term.
- Begin thesis write-up.
- Write up summary of all projects and experiments completed.
- Continue work on outstanding projects.

Easter Term

- Finalise projects.
- Full thesis outline.
- Continue write-up.

Summer Term

- Thesis write-up.

Bibliography

[1] NVIDIA Deep Learning Accelerator, . URL http://nvdla.org/.

[2] Traffic prediction with advanced Graph Neural Networks | DeepMind, . URL https://deepmind.com/blog/article/traffic-prediction-with-advanced-graph-neural-networks.

[3] UPMEM, . URL https://www.upmem.com/.

[4] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web - WWW '13*, pages 37–48, Rio de Janeiro, Brazil, 2013. ACM Press. ISBN 978-1-4503-2035-1. doi: 10.1145/2488388.2488393. URL http://dl.acm.org/citation.cfm?doid=2488388.2488393.

[5] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to Represent Programs with Graphs. *arXiv:1711.00740 [cs]*, May 2018. URL http://arxiv.org/abs/1711.00740. arXiv: 1711.00740.

[6] Miltiadis Allamanis, Earl T. Barr, Soline Ducousso, and Zheng Gao. Typilus: Neural Type Hints. *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 91–105, June 2020. doi: 10.1145/3385412.3385997. URL http://arxiv.org/abs/2004.10657. arXiv: 2004.10657.

[7] Anonymous. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. September 2020. URL https://openreview.net/forum?id=YicbFdNTTy.

[8] Junya Arai, Hiroaki Shiokawa, Takeshi Yamamuro, Makoto Onizuka, and Sotetsu Iwamura. Rabbit Order: Just-in-Time Parallel Reordering for Fast Graph Analysis. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 22–31, May 2016. doi: 10.1109/IPDPS.2016.110. ISSN: 1530-2075.

[9] Adam Auten, Matthew Tomei, and Rakesh Kumar. Hardware Acceleration of Graph Neural Networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*,

pages 1–6, San Francisco, CA, USA, July 2020. IEEE. ISBN 978-1-72811-085-1. doi: 10.1109/DAC18072.2020.9218751. URL https://ieeexplore.ieee.org/document/9218751/.

[10] Vignesh Balaji and Brandon Lucia. When is Graph Reordering an Optimization? Studying the Effect of Lightweight Graph Reordering Across Applications and Input Graphs. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 203–214, Raleigh, NC, September 2018. IEEE. ISBN 978-1-5386-6780-4. doi: 10.1109/IISWC.2018.8573478. URL https://ieeexplore.ieee.org/document/8573478/.

[11] Dominique Beaini, Saro Passaro, Vincent Létourneau, William L. Hamilton, Gabriele Corso, and Pietro Liò. Directional Graph Networks. *arXiv:2010.02863 [cs]*, October 2020. URL http://arxiv.org/abs/2010.02863. arXiv: 2010.02863.

[12] Mikhail Belkin and Partha Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 585–591. MIT Press, 2002. URL http://papers.nips.cc/paper/1961-laplacian-eigenmaps-and-spectral-techniques-for-embedding-and-clustering.pdf.

[13] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The Long-Document Transformer. *arXiv:2004.05150 [cs]*, April 2020. URL http://arxiv.org/abs/2004.05150. arXiv: 2004.05150.

[14] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv:1308.3432 [cs]*, August 2013. URL http://arxiv.org/abs/1308.3432. arXiv: 1308.3432.

[15] Michele Bevilacqua and Roberto Navigli. Breaking Through the 80% Glass Ceiling: Raising the State of the Art in Word Sense Disambiguation by Incorporating Knowledge Graph Information. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2854–2864, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.255. URL https://www.aclweb.org/anthology/2020.acl-main.255.

[16] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the State of Neural Network Pruning? page 18.

[17] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. *arXiv:1011.5425 [physics]*, October 2011. URL `http://arxiv.org/abs/1011.5425`. arXiv: 1011.5425.

[18] Dennis D. Boos. Using Extreme Value Theory to Estimate Large Percentiles. *Technometrics*, 26(1):33–39, February 1984. ISSN 0040-1706. doi: 10.1080/00401706.1984.10487919. URL `https://www.tandfonline.com/doi/abs/10.1080/00401706.1984.10487919`. Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/00401706.1984.10487919.

[19] Xavier Bresson and Thomas Laurent. Residual Gated Graph ConvNets. *arXiv:1711.07553 [cs, stat]*, April 2018. URL `http://arxiv.org/abs/1711.07553`. arXiv: 1711.07553.

[20] Andrew Brock, Theodore Lim, J. M. Ritchie, and Nick Weston. SMASH: One-Shot Model Architecture Search through HyperNetworks. *arXiv:1708.05344 [cs]*, August 2017. URL `http://arxiv.org/abs/1708.05344`. arXiv: 1708.05344.

[21] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]*, July 2020. URL `http://arxiv.org/abs/2005.14165`. arXiv: 2005.14165.

[22] Gino Brunner, Yang Liu, Damián Pascual, Oliver Richter, Massimiliano Ciaramita, and Roger Wattenhofer. On Identifiability in Transformers. *arXiv:1908.04211 [cs]*, February 2020. URL `http://arxiv.org/abs/1908.04211`. arXiv: 1908.04211.

[23] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. *arXiv:1812.00332 [cs, stat]*, February 2019. URL `http://arxiv.org/abs/1812.00332`. arXiv: 1812.00332.

[24] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-All: Train One Network and Specialize it for Efficient Deployment. *arXiv:1908.09791 [cs, stat]*, April 2020. URL `http://arxiv.org/abs/1908.09791`. arXiv: 1908.09791.

[25] Qi Cao, Huawei Shen, Jinhua Gao, Bingzheng Wei, and Xueqi Cheng. Popularity Prediction on Social Platforms with Coupled Graph Neural Networks. In *Proceedings*

of the 13th International Conference on Web Search and Data Mining, pages 70–78, Houston TX USA, January 2020. ACM. ISBN 978-1-4503-6822-3. doi: 10.1145/3336191.3371834. URL https://dl.acm.org/doi/10.1145/3336191.3371834.

[26] Shaosheng Cao, Wei Lu, and Qiongkai Xu. GraRep: Learning Graph Representations with Global Structural Information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 891–900, New York, NY, USA, October 2015. Association for Computing Machinery. ISBN 978-1-4503-3794-6. doi: 10.1145/2806416.2806512. URL https://doi.org/10.1145/2806416.2806512.

[27] M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, and M. Federico. Report on the 11 th IWSLT Evaluation Campaign , IWSLT 2014, 2015. URL /paper/Report-on-the-11-th-IWSLT-Evaluation-Campaign-%2C-Cettolo-Niehues/81aace0e90c6a962059b117c24db0d856f340f41.

[28] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. *arXiv:1909.03211 [cs, stat]*, November 2019. URL http://arxiv.org/abs/1909.03211. arXiv: 1909.03211.

[29] Hongming Chen, Ola Engkvist, Yinhai Wang, Marcus Olivecrona, and Thomas Blaschke. The rise of deep learning in drug discovery. *Drug Discovery Today*, 23 (6):1241–1250, June 2018. ISSN 1359-6446. doi: 10.1016/j.drudis.2018.01.039. URL http://www.sciencedirect.com/science/article/pii/S1359644617303598.

[30] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and Deep Graph Convolutional Networks. *arXiv:2007.02133 [cs, stat]*, July 2020. URL http://arxiv.org/abs/2007.02133. arXiv: 2007.02133.

[31] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to Optimize Tensor Programs. *arXiv:1805.08166 [cs, stat]*, January 2019. URL http://arxiv.org/abs/1805.08166. arXiv: 1805.08166.

[32] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, ASPLOS '14, pages 269–284, New York, NY, USA, February 2014. Association for Computing Machinery. ISBN 978-1-4503-2305-5. doi: 10.1145/2541940.2541967. URL https://doi.org/10.1145/2541940.2541967.

[33] Xiaobing Chen, Yuke Wang, Xinfeng Xie, Xing Hu, Abanti Basak, Ling Liang, Mingyu Yan, Lei Deng, Yufei Ding, Zidong Du, Yunji Chen, and Yuan Xie. Rubik: A Hierarchical Architecture for Efficient Graph Learning. *arXiv:2009.12495 [cs]*, September 2020. URL `http://arxiv.org/abs/2009.12495`. arXiv: 2009.12495.

[34] Xuhao Chen. Escoin: Efficient Sparse Convolutional Neural Network Inference on GPUs. *arXiv:1802.10280 [cs]*, April 2019. URL `http://arxiv.org/abs/1802.10280`. arXiv: 1802.10280.

[35] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, January 2017. ISSN 0018-9200, 1558-173X. doi: 10.1109/JSSC.2016.2616357. URL `http://ieeexplore.ieee.org/document/7738524/`.

[36] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *arXiv:1807.07928 [cs]*, May 2019. URL `http://arxiv.org/abs/1807.07928`. arXiv: 1807.07928.

[37] Ke Cheng, Yifan Zhang, Xiangyu He, Weihan Chen, Jian Cheng, and Hanqing Lu. Skeleton-Based Action Recognition With Shift Graph Convolutional Network. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 180–189, Seattle, WA, USA, June 2020. IEEE. ISBN 978-1-72817-168-5. doi: 10.1109/CVPR42600.2020.00026. URL `https://ieeexplore.ieee.org/document/9157077/`.

[38] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating Long Sequences with Sparse Transformers. *arXiv:1904.10509 [cs, stat]*, April 2019. URL `http://arxiv.org/abs/1904.10509`. arXiv: 1904.10509.

[39] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking Attention with Performers. *arXiv:2009.14794 [cs, stat]*, September 2020. URL `http://arxiv.org/abs/2009.14794`. arXiv: 2009.14794.

[40] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the Relationship between Self-Attention and Convolutional Layers. *arXiv:1911.03584 [cs, stat]*, January 2020. URL `http://arxiv.org/abs/1911.03584`. arXiv: 1911.03584.

[41] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 978-0-262-03384-8.

[42] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal Neighbourhood Aggregation for Graph Nets. *arXiv:2004.05718 [cs, stat]*, June 2020. URL http://arxiv.org/abs/2004.05718. arXiv: 2004.05718.

[43] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv:1602.02830 [cs]*, March 2016. URL http://arxiv.org/abs/1602.02830. arXiv: 1602.02830.

[44] Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alexander J Smola, and Le Song. Learning Steady-States of Iterative Algorithms over Graphs. page 9.

[45] Yaniv David, Uri Alon, and Eran Yahav. Neural Reverse Engineering of Stripped Binaries. *arXiv:1902.09122 [cs, stat]*, May 2020. URL http://arxiv.org/abs/1902.09122. arXiv: 1902.09122.

[46] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, CA, 2004.

[47] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *arXiv:1606.09375 [cs, stat]*, February 2017. URL http://arxiv.org/abs/1606.09375. arXiv: 1606.09375.

[48] Łukasz Dudziak, Thomas Chau, Mohamed S. Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D. Lane. BRP-NAS: Prediction-based NAS using GCNs. *arXiv:2007.08668 [cs, eess, stat]*, August 2020. URL http://arxiv.org/abs/2007.08668. arXiv: 2007.08668.

[49] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking Graph Neural Networks. *arXiv:2003.00982 [cs, stat]*, July 2020. URL http://arxiv.org/abs/2003.00982. arXiv: 2003.00982.

[50] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast Sparse ConvNets. *arXiv:1911.09723 [cs]*, November 2019. URL http://arxiv.org/abs/1911.09723. arXiv: 1911.09723.

[51] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the Lottery: Making All Tickets Winners. *arXiv:1911.11134 [cs, stat]*, July 2020. URL http://arxiv.org/abs/1911.11134. arXiv: 1911.11134.

[52] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. Training with Quantization Noise for Extreme

Model Compression. *arXiv:2004.07320 [cs, stat]*, April 2020. URL http://arxiv.org/abs/2004.07320. arXiv: 2004.07320.

[53] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph Neural Networks for Social Recommendation. *arXiv:1902.07243 [cs]*, November 2019. URL http://arxiv.org/abs/1902.07243. arXiv: 1902.07243.

[54] Boyuan Feng, Yuke Wang, Xu Li, Shu Yang, Xueqiao Peng, and Yufei Ding. SGQuant: Squeezing the Last Bit on Graph Neural Networks with Specialized Quantization. *arXiv:2007.05100 [cs, stat]*, September 2020. URL http://arxiv.org/abs/2007.05100. arXiv: 2007.05100.

[55] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Pruning Neural Networks at Initialization: Why are We Missing the Mark? *arXiv:2009.08576 [cs, stat]*, September 2020. URL http://arxiv.org/abs/2009.08576. arXiv: 2009.08576.

[56] Adi Fuchs and David Wentzlaff. The Accelerator Wall: Limits of Chip Specialization. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–14, February 2019. doi: 10.1109/HPCA.2019.00023. ISSN: 2378-203X.

[57] Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse GPU Kernels for Deep Learning. *arXiv:2006.10901 [cs, stat]*, August 2020. URL http://arxiv.org/abs/2006.10901. arXiv: 2006.10901.

[58] Claudio Gallicchio and Alessio Micheli. Graph Echo State Networks. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2010. doi: 10.1109/IJCNN.2010.5596796. ISSN: 2161-4407.

[59] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. GraphNAS: Graph Neural Architecture Search with Reinforcement Learning. *arXiv:1904.09981 [cs, stat]*, August 2019. URL http://arxiv.org/abs/1904.09981. arXiv: 1904.09981 version: 2.

[60] Liuhao Ge, Zhou Ren, Yuncheng Li, Zehao Xue, Yingying Wang, Jianfei Cai, and Junsong Yuan. 3D Hand Shape and Pose Estimation from a Single RGB Image. *arXiv:1903.00812 [cs]*, April 2019. URL http://arxiv.org/abs/1903.00812. arXiv: 1903.00812.

[61] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv:1811.12231*

[cs, q-bio, stat], January 2019. URL http://arxiv.org/abs/1811.12231. arXiv: 1811.12231.

[62] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steve Reinhardt, and Martin Herbordt. AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing. *arXiv:1908.10834 [cs]*, September 2020. URL http://arxiv.org/abs/1908.10834. arXiv: 1908.10834.

[63] Georgios Georgiadis. Accelerating Convolutional Neural Networks via Activation Map Compression. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7078–7088, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00725. URL https://ieeexplore.ieee.org/document/8953659/.

[64] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for Quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1263–1272, Sydney, NSW, Australia, August 2017. JMLR.org.

[65] Joseph E Gonzalez, Yucheng Low, and Haijie Gu. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. page 14.

[66] Saket Gurukar, Priyesh Vijayan, Aakash Srinivasan, Goonmeet Bajaj, Chen Cai, Moniba Keymanesh, Saravana Kumar, Pranav Maneriker, Anasua Mitra, Vedang Patel, Balaraman Ravindran, and Srinivasan Parthasarathy. Network Representation Learning: Consolidation and Renewed Bearing. *arXiv:1905.00987 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1905.00987. arXiv: 1905.00987 version: 2.

[67] Ameer Haj-Ali, Qijing Huang, William Moses, John Xiang, Ion Stoica, Krste Asanovic, and John Wawrzynek. AutoPhase: Compiler Phase-Ordering for High Level Synthesis with Deep Reinforcement Learning. *arXiv:1901.04615 [cs]*, April 2019. URL http://arxiv.org/abs/1901.04615. arXiv: 1901.04615.

[68] Tae Jun Ham, Lisa Wu, Narayanan Sundaram, Nadathur Satish, and Margaret Martonosi. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13, October 2016. doi: 10.1109/MICRO.2016.7783759.

[69] William L Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. page 23.

[70] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. *arXiv:1706.02216 [cs, stat]*, September 2018. URL http://arxiv.org/abs/1706.02216. arXiv: 1706.02216.

[71] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. URL http://arxiv.org/abs/1512.03385. arXiv: 1512.03385.

[72] David Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. In David Hilbert, editor, *Dritter Band: Analysis · Grundlagen der Mathematik · Physik Verschiedenes: Nebst Einer Lebensgeschichte*, pages 1–2. Springer, Berlin, Heidelberg, 1935. ISBN 978-3-662-38452-7. doi: 10.1007/978-3-662-38452-7_1. URL https://doi.org/10.1007/978-3-662-38452-7_1.

[73] Valentin Hofmann, Hinrich Schütze, and Janet Pierrehumbert. A Graph Auto-encoder Model of Derivational Morphology. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1127–1138, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.106. URL https://www.aclweb.org/anthology/2020.acl-main.106.

[74] Mark Horowitz. 1.1 Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, February 2014. doi: 10.1109/ISSCC.2014.6757323. ISSN: 2376-8606.

[75] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*, April 2017. URL http://arxiv.org/abs/1704.04861. arXiv: 1704.04861.

[76] Patrik O. Hoyer. Non-negative Matrix Factorization with Sparseness Constraints. *The Journal of Machine Learning Research*, 5:1457–1469, December 2004. ISSN 1532-4435.

[77] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for Pre-training Graph Neural Networks. *arXiv:1905.12265 [cs, stat]*, February 2020. URL http://arxiv.org/abs/1905.12265. arXiv: 1905.12265.

[78] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv:1602.07360 [cs]*, November 2016. URL http://arxiv.org/abs/1602.07360. arXiv: 1602.07360.

[79] Forrest N. Iandola, Albert E. Shaw, Ravi Krishna, and Kurt W. Keutzer. SqueezeBERT: What can computer vision teach NLP about efficient neural networks? *arXiv:2006.11316 [cs]*, June 2020. URL http://arxiv.org/abs/2006.11316. arXiv: 2006.11316.

[80] Sarthak Jain and Byron C. Wallace. Attention is not Explanation. *arXiv:1902.10186 [cs]*, May 2019. URL http://arxiv.org/abs/1902.10186. arXiv: 1902.10186.

[81] Steven A. Janowsky. Pruning versus clipping in neural networks. *Physical Review A*, 39(12):6600–6603, June 1989. doi: 10.1103/PhysRevA.39.6600. URL https://link.aps.org/doi/10.1103/PhysRevA.39.6600. Publisher: American Physical Society.

[82] Zhihao Jia, Yongkee Kwon, Galen Shipman, Pat McCormick, Mattan Erez, and Alex Aiken. A distributed multi-GPU system for fast graph processing. *Proceedings of the VLDB Endowment*, 11(3):297–310, November 2017. ISSN 2150-8097. doi: 10.14778/3157794.3157799. URL https://dl.acm.org/doi/10.14778/3157794.3157799.

[83] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond Data and Model Parallelism for Deep Neural Networks. *arXiv:1807.05358 [cs]*, July 2018. URL http://arxiv.org/abs/1807.05358. arXiv: 1807.05358.

[84] Zhihao Jia, Sina Lin, Rex Ying, Jiaxuan You, Jure Leskovec, and Alex Aiken. Redundancy-Free Computation Graphs for Graph Neural Networks. *arXiv:1906.03707 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1906.03707. arXiv: 1906.03707.

[85] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with Roc. *Proceedings of Machine Learning and Systems*, 2, March 2020. URL https://proceedings.mlsys.org/paper/2020/hash/fe9fc289c3ff0af142b6d3bead98a923-Abstract.html.

[86] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller,

Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12, June 2017. doi: 10.1145/3079856.3080246.

[87] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, Jiyan Yang, Jongsoo Park, Alexander Heinecke, Evangelos Georganas, Sudarshan Srinivasan, Abhisek Kundu, Misha Smelyanskiy, Bharat Kaul, and Pradeep Dubey. A Study of BFLOAT16 for Deep Learning Training. *arXiv:1905.12322 [cs, stat]*, June 2019. URL `http://arxiv.org/abs/1905.12322`. arXiv: 1905.12322.

[88] Konstantinos I. Karantasis, Andrew Lenharth, Donald Nguyen, Marás J. Garzarán, and Keshav Pingali. Parallelization of Reordering Algorithms for Bandwidth and Wavefront Reduction. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 921–932, November 2014. doi: 10.1109/SC.2014.80. ISSN: 2167-4337.

[89] E.D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, June 1990. ISSN 1941-0093. doi: 10.1109/72.80236. Conference Name: IEEE Transactions on Neural Networks.

[90] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. *arXiv:2006.16236 [cs, stat]*, August 2020. URL `http://arxiv.org/abs/2006.16236`. arXiv: 2006.16236.

[91] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv:1609.02907 [cs, stat]*, February 2017. URL `http://arxiv.org/abs/1609.02907`. arXiv: 1609.02907.

[92] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The Efficient Transformer. *arXiv:2001.04451 [cs, stat]*, February 2020. URL `http://arxiv.org/abs/2001.04451`. arXiv: 2001.04451.

[93] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv:1806.08342 [cs, stat]*, June 2018. URL `http://arxiv.org/abs/1806.08342`. arXiv: 1806.08342.

[94] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[95] P. A. Kulkarni, D. B. Whalley, G. S. Tyson, and J. W. Davidson. Exhaustive optimization phase order space exploration. In *International Symposium on Code Generation and Optimization (CGO'06)*, pages 13 pp.–318, March 2006. doi: 10.1109/CGO.2006.15.

[96] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Bill Nell, Nir Shavit, and Dan Alistarh. Inducing and Exploiting Activation Sparsity for Fast Neural Network Inference. page 11.

[97] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. GraphChi: Large-Scale Graph Computation on Just a PC. page 17.

[98] Kartik Lakhotia, Shreyas Singapura, Rajgopal Kannan, and Viktor Prasanna. ReCALL: Reordered Cache Aware Locality Based Graph Processing. In *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*, pages 273–282, December 2017. doi: 10.1109/HiPC.2017.00039.

[99] Dominique Lasalle and George Karypis. Multi-threaded Graph Partitioning. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 225–236, May 2013. doi: 10.1109/IPDPS.2013.50. ISSN: 1530-2075.

[100] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, CGO '04, page 75, USA, March 2004. IEEE Computer Society. ISBN 978-0-7695-2102-2.

[101] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. SNIP: Single-shot Network Pruning based on Connection Sensitivity. *arXiv:1810.02340 [cs]*, February 2019. URL `http://arxiv.org/abs/1810.02340`. arXiv: 1810.02340.

[102] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 3538–3545. Association for the Advancement of Artificial Intelligence, February 2018. ISBN 978-1-57735-800-8. URL https://www.research-collection.ethz.ch/handle/20.500.11850/368499. Accepted: 2019-10-07T09:59:39Z.

[103] Sheng Li, Jongsoo Park, and Ping Tak Peter Tang. Enabling Sparse Winograd Convolution by Native Pruning. *arXiv:1702.08597 [cs]*, October 2017. URL http://arxiv.org/abs/1702.08597. arXiv: 1702.08597.

[104] Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. Gated Graph Sequence Neural Networks. November 2015. URL https://www.microsoft.com/en-us/research/publication/gated-graph-sequence-neural-networks/.

[105] Vasileios Lioutas and Yuhong Guo. Time-aware Large Kernel Convolutions. *arXiv:2002.03184 [cs, stat]*, June 2020. URL http://arxiv.org/abs/2002.03184. arXiv: 2002.03184.

[106] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. *arXiv:1806.09055 [cs, stat]*, April 2019. URL http://arxiv.org/abs/1806.09055. arXiv: 1806.09055.

[107] Xiaodong Liu, Kevin Duh, Liyuan Liu, and Jianfeng Gao. Very Deep Transformers for Neural Machine Translation. *arXiv:2008.07772 [cs]*, August 2020. URL http://arxiv.org/abs/2008.07772. arXiv: 2008.07772.

[108] Yozen Liu, Xiaolin Shi, Lucas Pierce, and Xiang Ren. Characterizing and Forecasting User Engagement with In-app Action Graph: A Case Study of Snapchat. *arXiv:1906.00355 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1906.00355. arXiv: 1906.00355.

[109] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian Compression for Deep Learning. *arXiv:1705.08665 [cs, stat]*, November 2017. URL http://arxiv.org/abs/1705.08665. arXiv: 1705.08665.

[110] Yucheng Low, Joseph E. Gonzalez, Aapo Kyrola, Danny Bickson, Carlos E. Guestrin, and Joseph Hellerstein. GraphLab: A New Framework For Parallel Machine Learning. *UAI 2010*, August 2014. URL http://arxiv.org/abs/1408.2041. arXiv: 1408.2041.

[111] Arm Ltd. Ethos-U55 Machine Learning Processor. URL https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u55.

[112] Yi-Ju Lu and Cheng-Te Li. GCAN: Graph-aware Co-Attention Networks for Explainable Fake News Detection on Social Media. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 505–514, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.48. URL https://www.aclweb.org/anthology/2020.acl-main.48.

[113] Ying Luo and Hai Zhao. Bipartite Flat-Graph Network for Nested Named Entity Recognition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6408–6418, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.571. URL https://www.aclweb.org/anthology/2020.acl-main.571.

[114] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 international conference on Management of data*, pages 135–146, New York, NY, USA, 2010. URL http://doi.acm.org/10.1145/1807167.1807184.

[115] Swetha Mandava, Szymon Migacz, and Alex Fit Florea. Pay Attention when Required. *arXiv:2009.04534 [cs]*, September 2020. URL http://arxiv.org/abs/2009.04534. arXiv: 2009.04534.

[116] Frank McSherry, Michael Isard, and Derek G. Murray. Scalability! but at what cost? In *Proceedings of the 15th USENIX conference on Hot Topics in Operating Systems*, HOTOS'15, page 14, USA, May 2015. USENIX Association.

[117] Sachin Mehta, Marjan Ghazvininejad, Srinivasan Iyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. DeLighT: Very Deep and Light-weight Transformer. *arXiv:2008.00623 [cs]*, August 2020. URL http://arxiv.org/abs/2008.00623. arXiv: 2008.00623.

[118] Charith Mendis, Cambridge Yang, Yewen Pu, Dr.Saman Amarasinghe, and Michael Carbin. Compiler Auto-Vectorization with Imitation Learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 14625–14635. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9604-compiler-auto-vectorization-with-imitation-learning.pdf.

[119] Paul Michel, Omer Levy, and Graham Neubig. Are Sixteen Heads Really Better than One? *arXiv:1905.10650 [cs]*, November 2019. URL http://arxiv.org/abs/1905.10650. arXiv: 1905.10650.

[120] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed Precision Training. *arXiv:1710.03740 [cs, stat]*, February 2018. URL http://arxiv.org/abs/1710.03740. arXiv: 1710.03740.

[121] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device Placement Optimization with Reinforcement Learning. *arXiv:1706.04972 [cs]*, June 2017. URL http://arxiv.org/abs/1706.04972. arXiv: 1706.04972.

[122] Michael C Mozer and Paul Smolensky. Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 107–115. Morgan-Kaufmann, 1989. URL http://papers.nips.cc/paper/119-skeletonization-a-technique-for-trimming-the-fat-from-a-network-via-relev pdf.

[123] MICHAEL C. MOZER and PAUL SMOLENSKY. Using Relevance to Reduce Network Size Automatically. *Connection Science*, 1(1):3–16, January 1989. ISSN 0954-0091. doi: 10.1080/09540098908915626. URL https://doi.org/10.1080/09540098908915626. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/09540098908915626.

[124] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. page 8.

[125] Hoang NT and Takanori Maehara. Revisiting Graph Neural Networks: All We Have is Low-Pass Filters. *arXiv:1905.09550 [cs, math, stat]*, May 2019. URL http://arxiv.org/abs/1905.09550. arXiv: 1905.09550.

[126] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric Transitivity Preserving Graph Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1105–1114, San Francisco California USA, August 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939751. URL https://dl.acm.org/doi/10.1145/2939672.2939751.

[127] Larry Page, Sergey Brin, R. Motwani, and T. Winograd. *The PageRank Citation Ranking: Bringing Order to the Web*. 1998.

[128] Subhankar Pal, Jonathan Beaumont, Dong-Hyeon Park, Aporva Amarnath, Siying Feng, Chaitali Chakrabarti, Hun-Seok Kim, David Blaauw, Trevor Mudge,

and Ronald Dreslinski. OuterSPACE: An Outer Product Based Sparse Matrix Multiplication Accelerator. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 724–736, February 2018. doi: 10.1109/HPCA.2018.00067. ISSN: 2378-203X.

[129] Boxiao Pan, Haoye Cai, De-An Huang, Kuan-Hui Lee, Adrien Gaidon, Ehsan Adeli, and Juan Carlos Niebles. Spatio-Temporal Graph for Video Captioning With Knowledge Distillation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10867–10876, Seattle, WA, USA, June 2020. IEEE. ISBN 978-1-72817-168-5. doi: 10.1109/CVPR42600.2020.01088. URL `https://ieeexplore.ieee.org/document/9156988/`.

[130] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL `https://www.aclweb.org/anthology/P02-1040`.

[131] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. *arXiv:1708.04485 [cs]*, May 2017. URL `http://arxiv.org/abs/1708.04485`. arXiv: 1708.04485.

[132] Jongsoo Park, Sheng Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. Faster CNNs with Direct Sparse Convolutions and Guided Pruning. *arXiv:1608.01409 [cs]*, July 2017. URL `http://arxiv.org/abs/1608.01409`. arXiv: 1608.01409.

[133] Maurice Peemen, Arnaud A. A. Setio, Bart Mesman, and Henk Corporaal. Memory-centric accelerator design for Convolutional Neural Networks. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 13–19, October 2013. doi: 10.1109/ICCD.2013.6657019. ISSN: 1063-6404.

[134] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient Neural Architecture Search via Parameter Sharing. *arXiv:1802.03268 [cs, stat]*, February 2018. URL `http://arxiv.org/abs/1802.03268`. arXiv: 1802.03268.

[135] Ofir Press, Noah A. Smith, and Omer Levy. Improving Transformer Models by Reordering their Sublayers. In *Proceedings of the 58th Annual Meeting of the*

Association for Computational Linguistics, pages 2996–3005, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.270. URL https://www.aclweb.org/anthology/2020.acl-main.270.

[136] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. DeepInf: Social Influence Prediction with Deep Learning. *arXiv:1807.05560 [cs]*, July 2018. doi: 10.1145/3219819.3220077. URL http://arxiv.org/abs/1807.05560. arXiv: 1807.05560.

[137] Alex Renda, Yishen Chen, Charith Mendis, and Michael Carbin. DiffTune: Optimizing CPU Simulator Parameters with Learned Differentiable Surrogates. *arXiv:2010.04017 [cs]*, October 2020. URL http://arxiv.org/abs/2010.04017. arXiv: 2010.04017.

[138] Minsoo Rhu, Mike O'Connor, Niladrish Chatterjee, Jeff Pool, and Stephen W. Keckler. Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks. *arXiv:1705.01626 [cs]*, May 2017. URL http://arxiv.org/abs/1705.01626. arXiv: 1705.01626.

[139] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. *arXiv:1907.10903 [cs, stat]*, March 2020. URL http://arxiv.org/abs/1907.10903. arXiv: 1907.10903.

[140] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. X-Stream: edge-centric graph processing using streaming partitions. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles - SOSP '13*, pages 472–488, Farminton, Pennsylvania, 2013. ACM Press. ISBN 978-1-4503-2388-8. doi: 10.1145/2517349.2522740. URL http://dl.acm.org/citation.cfm?doid=2517349.2522740.

[141] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. SuperGlue: Learning Feature Matching With Graph Neural Networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4937–4946, Seattle, WA, USA, June 2020. IEEE. ISBN 978-1-72817-168-5. doi: 10.1109/CVPR42600.2020.00499. URL https://ieeexplore.ieee.org/document/9157489/.

[142] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, January 2009. ISSN 1941-0093. doi: 10.1109/TNN.2008.2005605. Conference Name: IEEE Transactions on Neural Networks.

[143] Weijing Shi, Ragunathan, and Rajkumar. Point-GNN: Graph Neural Network for 3D Object Detection in a Point Cloud. *arXiv:2003.01251 [cs]*, March 2020. URL http://arxiv.org/abs/2003.01251. arXiv: 2003.01251.

[144] David I Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, May 2013. ISSN 1558-0792. doi: 10.1109/MSP.2012.2235192. Conference Name: IEEE Signal Processing Magazine.

[145] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the Bit Goes Down: Revisiting the Quantization of Neural Networks. *arXiv:1907.05686 [cs]*, December 2019. URL http://arxiv.org/abs/1907.05686. arXiv: 1907.05686.

[146] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive Attention Span in Transformers. *arXiv:1905.07799 [cs, stat]*, August 2019. URL http://arxiv.org/abs/1905.07799. arXiv: 1905.07799.

[147] Narayanan Sundaram, Nadathur Satish, Md Mostofa Ali Patwary, Subramanya R. Dulloor, Michael J. Anderson, Satya Gautam Vadlamudi, Dipankar Das, and Pradeep Dubey. GraphMat: high performance graph analytics made productive. *Proceedings of the VLDB Endowment*, 8(11):1214–1225, July 2015. ISSN 2150-8097. doi: 10.14778/2809974.2809983. URL https://doi.org/10.14778/2809974.2809983.

[148] Shyam A. Tailor, Jagmohan Chauhan, and Cecilia Mascolo. A first step towards on-device monitoring of body sounds in the wild. In *WellComp Workshop, UbiComp/ISWC*, 2020.

[149] Shyam A. Tailor, Javier Fernandez-Marques, and Nicholas D. Lane. Degree-Quant: Quantization-Aware Training for Graph Neural Networks. In *Graph Representation Learning and Beyond Workshop, ICML*, 2020.

[150] Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv:2006.05467 [cond-mat, q-bio, stat]*, June 2020. URL http://arxiv.org/abs/2006.05467. arXiv: 2006.05467.

[151] Daniel Tarlow, Subhodeep Moitra, Andrew Rice, Zimin Chen, Pierre-Antoine Manzagol, Charles Sutton, and Edward Aftandilian. Learning to Fix Build Errors with Graph2Diff Neural Networks. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 19–20, Seoul Republic of

Korea, June 2020. ACM. ISBN 978-1-4503-7963-2. doi: 10.1145/3387940.3392181. URL https://dl.acm.org/doi/10.1145/3387940.3392181.

[152] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Synthesizer: Rethinking Self-Attention in Transformer Models. *arXiv:2005.00743 [cs]*, May 2020. URL http://arxiv.org/abs/2005.00743. arXiv: 2005.00743.

[153] Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso. The Computational Limits of Deep Learning. *arXiv:2007.05558 [cs, stat]*, July 2020. URL http://arxiv.org/abs/2007.05558. arXiv: 2007.05558.

[154] Catherine Tong, Shyam A. Tailor, and Nicholas D. Lane. Are Accelerometers for Activity Recognition a Dead-end? In *HotMobile*, 2020.

[155] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

[156] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *arXiv:1710.10903 [cs, stat]*, February 2018. URL http://arxiv.org/abs/1710.10903. arXiv: 1710.10903.

[157] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, Peter Vajda, and Joseph E. Gonzalez. FBNetV2: Differentiable Neural Architecture Search for Spatial and Channel Dimensions. *arXiv:2004.05565 [cs]*, April 2020. URL http://arxiv.org/abs/2004.05565. arXiv: 2004.05565.

[158] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking Winning Tickets Before Training by Preserving Gradient Flow. *arXiv:2002.07376 [cs, stat]*, August 2020. URL http://arxiv.org/abs/2002.07376. arXiv: 2002.07376.

[159] Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, Xiangjian He, Yiguang Lin, and Xuemin Lin. Binarized Graph Neural Network. *arXiv:2004.11147 [cs, stat]*, April 2020. URL http://arxiv.org/abs/2004.11147. arXiv: 2004.11147.

[160] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-Aware Automated Quantization With Mixed Precision. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8604–8612, Long Beach,

CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00881. URL https://ieeexplore.ieee.org/document/8954415/.

[161] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training Deep Neural Networks with 8-bit Floating Point Numbers. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7675–7684. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7994-training-deep-neural-networks-with-8-bit-floating-point-numbers.pdf.

[162] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-Attention with Linear Complexity. *arXiv:2006.04768 [cs, stat]*, June 2020. URL http://arxiv.org/abs/2006.04768. arXiv: 2006.04768.

[163] Hao Wei, Jeffrey Xu Yu, Can Lu, and Xuemin Lin. Speedup Graph Processing by Graph Ordering. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 1813–1828, New York, NY, USA, June 2016. Association for Computing Machinery. ISBN 978-1-4503-3531-7. doi: 10.1145/2882903.2915220. URL https://doi.org/10.1145/2882903.2915220.

[164] Sarah Wiegreffe and Yuval Pinter. Attention is not not Explanation. *arXiv:1908.04626 [cs]*, September 2019. URL http://arxiv.org/abs/1908.04626. arXiv: 1908.04626.

[165] Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. Pay Less Attention with Lightweight and Dynamic Convolutions. *arXiv:1901.10430 [cs]*, February 2019. URL http://arxiv.org/abs/1901.10430. arXiv: 1901.10430.

[166] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr., Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying Graph Convolutional Networks. *arXiv:1902.07153 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1902.07153. arXiv: 1902.07153.

[167] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2020. ISSN 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2020.2978386. URL http://arxiv.org/abs/1901.00596. arXiv: 1901.00596.

[168] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? *arXiv:1810.00826 [cs, stat]*, February 2019. URL http://arxiv.org/abs/1810.00826. arXiv: 1810.00826.

[169] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. Grid-GCN for Fast and Scalable Point Cloud Learning. *arXiv:1912.02984 [cs]*, March 2020. URL http://arxiv.org/abs/1912.02984. arXiv: 1912.02984.

[170] Bencheng Yan, Chaokun Wang, Gaoyang Guo, and Yunkai Lou. TinyGNN: Learning Efficient Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1848–1856, Virtual Event CA USA, August 2020. ACM. ISBN 978-1-4503-7998-4. doi: 10.1145/3394486.3403236. URL https://dl.acm.org/doi/10.1145/3394486.3403236.

[171] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. HyGCN: A GCN Accelerator with Hybrid Architecture. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 15–29, February 2020. doi: 10.1109/HPCA47549.2020.00012. ISSN: 2378-203X.

[172] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards Reproducible Neural Architecture Search. *arXiv:1902.09635 [cs, stat]*, May 2019. URL http://arxiv.org/abs/1902.09635. arXiv: 1902.09635.

[173] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, July 2018. doi: 10.1145/3219819.3219890. URL http://arxiv.org/abs/1806.01973. arXiv: 1806.01973.

[174] Weiqiu You, Simeng Sun, and Mohit Iyyer. Hard-Coded Gaussian Attention for Neural Machine Translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7689–7700, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.687. URL https://www.aclweb.org/anthology/2020.acl-main.687.

[175] Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J. Reddi, and Sanjiv Kumar. Are Transformers universal approximators of sequence-to-sequence functions? *arXiv:1912.10077 [cs, stat]*, February 2020. URL http://arxiv.org/abs/1912.10077. arXiv: 1912.10077.

[176] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for Longer Sequences. *arXiv:2007.14062 [cs, stat]*, July 2020. URL http://arxiv.org/abs/2007.14062. arXiv: 2007.14062.

[177] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph Sampling Based Inductive Learning Method. *arXiv:1907.04931 [cs, stat]*, February 2020. URL http://arxiv.org/abs/1907.04931. arXiv: 1907.04931.

[178] 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, pages 161–170, New York, NY, USA, February 2015. Association for Computing Machinery. ISBN 978-1-4503-3315-3. doi: 10.1145/2684746.2689060. URL https://doi.org/10.1145/2684746.2689060.

[179] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, Salt Lake City, UT, June 2018. IEEE. ISBN 978-1-5386-6420-9. doi: 10.1109/CVPR.2018.00716. URL https://ieeexplore.ieee.org/document/8578814/.

[180] Xikun Zhang, Chang Xu, and Dacheng Tao. Context Aware Graph Convolution for Skeleton-Based Action Recognition. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14321–14330, Seattle, WA, USA, June 2020. IEEE. ISBN 978-1-72817-168-5. doi: 10.1109/CVPR42600.2020.01434. URL https://ieeexplore.ieee.org/document/9156373/.

[181] Yunming Zhang, Vladimir Kiriansky, Charith Mendis, Saman Amarasinghe, and Matei Zaharia. Making caches work for graph analytics. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 293–302, December 2017. doi: 10.1109/BigData.2017.8257937.

[182] Zhekai Zhang, Hanrui Wang, Song Han, and William J. Dally. SpArch: Efficient Architecture for Sparse Matrix Multiplication. *arXiv:2002.08947 [cs]*, February 2020. URL http://arxiv.org/abs/2002.08947. arXiv: 2002.08947.

[183] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring Self-attention for Image Recognition. *arXiv:2004.13621 [cs]*, April 2020. URL http://arxiv.org/abs/2004.13621. arXiv: 2004.13621.

[184] Lingxiao Zhao and Leman Akoglu. PairNorm: Tackling Oversmoothing in GNNs. *arXiv:1909.12223 [cs, stat]*, February 2020. URL http://arxiv.org/abs/1909.12223. arXiv: 1909.12223.

[185] Yiren Zhao, Duo Wang, Daniel Bates, Robert Mullins, Mateja Jamnik, and Pietro Lio. Learned Low Precision Graph Neural Networks. *arXiv:2009.09232 [cs]*, September 2020. URL http://arxiv.org/abs/2009.09232. arXiv: 2009.09232.

[186] Yiren Zhao, Duo Wang, Xitong Gao, Robert Mullins, Pietro Lio, and Mateja Jamnik. Probabilistic Dual Network Architecture Search on Graphs. *arXiv:2003.09676 [cs, stat]*, March 2020. URL http://arxiv.org/abs/2003.09676. arXiv: 2003.09676.

[187] Wanjun Zhong, Jingjing Xu, Duyu Tang, Zenan Xu, Nan Duan, Ming Zhou, Jiahai Wang, and Jian Yin. Reasoning Over Semantic-Level Graph for Fact Checking. *arXiv:1909.03745 [cs]*, September 2019. URL http://arxiv.org/abs/1909.03745. arXiv: 1909.03745 version: 1.

[188] Wanjun Zhong, Duyu Tang, Zhangyin Feng, Nan Duan, Ming Zhou, Ming Gong, Linjun Shou, Daxin Jiang, Jiahai Wang, and Jian Yin. LogicalFactChecker: Leveraging Logical Operations for Fact Checking with Graph Module Network. *arXiv:2004.13659 [cs]*, April 2020. URL http://arxiv.org/abs/2004.13659. arXiv: 2004.13659.

[189] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. Auto-GNN: Neural Architecture Search of Graph Neural Networks. *arXiv:1909.03184 [cs, stat]*, September 2019. URL http://arxiv.org/abs/1909.03184. arXiv: 1909.03184.

[190] Kuangqi Zhou, Yanfei Dong, Wee Sun Lee, Bryan Hooi, Huan Xu, and Jiashi Feng. Effective Training Strategies for Deep Graph Neural Networks. *arXiv:2006.07107 [cs, stat]*, June 2020. URL http://arxiv.org/abs/2006.07107. arXiv: 2006.07107.

[191] Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Lin-Kit Wong, Peter Ma, Qiumin Xu, Azalia Mirhoseini, and James Laudon. A Single-Shot Generalized Device Placement for Large Dataflow Graphs. *IEEE Micro*, 40(5):26–36, September 2020. ISSN 1937-4143. doi: 10.1109/MM.2020.3015188. Conference Name: IEEE Micro.

[192] Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. *arXiv:1611.01578 [cs]*, February 2017. URL http://arxiv.org/abs/1611.01578. arXiv: 1611.01578.

Appendix A

Published Works

Are Accelerometers for Activity Recognition a Dead-end?

Catherine Tong[†], Shyam A. Tailor[†], Nicholas D. Lane[†◇]
[†]University of Oxford [◇]Samsung AI

ABSTRACT

Accelerometer-based (and by extension other inertial sensors) research for Human Activity Recognition (HAR) is a dead-end. This sensor does not offer enough information for us to progress in the core domain of HAR—to recognize everyday activities from sensor data. Despite continued and prolonged efforts in improving feature engineering and machine learning models, the activities that we can recognize reliably have only expanded slightly and many of the same flaws of early models are still present today. Instead of relying on acceleration data, we should instead consider modalities with much richer information—a logical choice are images. With the rapid advance in image sensing hardware and modelling techniques, we believe that a widespread adoption of image sensors will open many opportunities for accurate and robust inference across a wide spectrum of human activities.

In this paper, we make the case for imagers in place of accelerometers as the default sensor for human activity recognition. Our review of past works has led to the observation that progress in HAR had stalled, caused by our reliance on accelerometers. We further argue for the suitability of images for activity recognition by illustrating their richness of information and the marked progress in computer vision. Through a feasibility analysis, we find that deploying imagers and CNNs on device poses no substantial burden on modern mobile hardware. Overall, our work highlights the need to move away from accelerometers and calls for further exploration of using imagers for activity recognition.[1]

1 INTRODUCTION

For a long time, the default sensors for Human Activity Recognition (HAR) have been accelerometers—low-cost, low-power and compact sensors which provide motion-related information. In the past decades, waves of accelerometer-based (and by extension other inertial sensors) research have enabled HAR studies ranging from the classification of common activities to the objective assessment of their quality, as seen in exciting applications such as skill [21] and disease rehabilitation [27] assessment at scale. However, despite these success stories, we believe that accelerometers can lead us no further in achieving the primary task at the very core of HAR—to recognize activities reliably and robustly.

The reliable recognition of activities, first and foremost, requires data to be collected from information-rich and energy-efficient sensors. In 2004, Bao and Intille [4] published a landmark study to recognize daily activities from acceleration data. 15 years on, we have not moved far from identifying activities much more complex than 'walking' or 'running' with mobile sensors [33]. This slow process is not due to a lack of data (accelerometers are present in almost all smart devices), nor a lack of feature engineering and algorithmic innovation (a great variety of accelerometer-based HAR works exists, including deep learning solutions), but the quality of the sensor data itself.

It is time to rethink this default HAR sensor and move towards a modality with richer information, in order to identify more activities more robustly. While accelerometers capture motion-related information useful for distinguishing elementary activities, a significant portion of our activities are not characterised by their motions but by their precise context, e.g. social setting or objects of interaction. Recognising these activities unobtrusively is of great interest to many research communities such as psychology and health sciences but currently we are unable to do this with acceleration data. Moving forward, can we rely on such a sensor which inherently lacks the dimension to capture non-motion-based information? Through inspecting years of progress in accelerometer-based HAR research, our answer is no. Our over-reliance on accelerometer-based data is contributing to a bottleneck in inference performance, a constrained list of recognisable activity classes and alarming confusion errors. To overcome this bottleneck, we should instead adopt a sensing modality with much richer information—the most natural choice are *images*.

Against the backdrop of a rapid sophistication of both learning algorithms and sensor hardware, imagers—low-form factor visual sensors—stand as a strong candidate to replace accelerometers as the default sensor in a new wave of HAR research. Notably, the idea that images are information-rich is not new: both stationary and wearable cameras have been used in numerous studies for visual-based action recognition [35], and for providing labelable ground truth and additional contexts [10]; The superiority of cameras compared to other sensors, including accelerometers, for high-level activity recognition has also been argued empirically [24]. Critically, previous barriers to the adoption of imagers have now been overcome by substantial advancement seen in recent years: on one hand, powerful and efficient deep learning algorithms have been developed for image processing [14, 23, 32]; on the other hand, there are now miniature image sensors with low energy requirement [1]. Our feasibility analysis suggests that the cost, size and energy requirements of imagers are no longer barriers to their adoption.

What we are proposing, adopting imagers as default sensors for activity recognition, is not to reject the use of accelerometers in tangential problems such as those assessing the level or quality of activities. It is also not to dismiss a multi-modality scenario, although we view the multitude of common low-dimensional sensors (e.g. magnetometer, gyroscope) as having only a supplementary

[1]A version of this paper was accepted at The 21st International Workshop on Mobile Computing Systems and Applications (HotMobile) 2020.

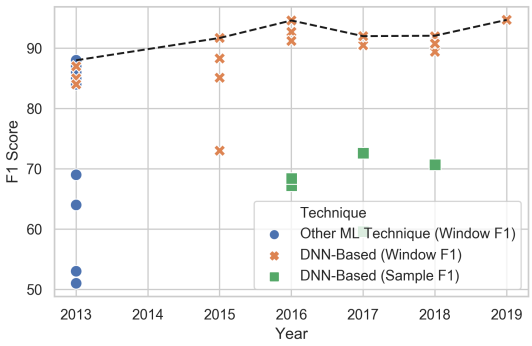


Figure 1: Opportunity gesture recognition, up to two scores plotted per publication per metric. We plot both the window-based F1 metric, and the sample-based F1 metric, which more recent publications prefer. We observe that deep learning has not yielded significant improvements to F1 score, despite its obvious success in other fields. Sample-based F1, believed to be a more accurate proxy to performance, is still not good enough to accurately disambiguate different activities that a user might perform.

role supporting the rich information collected by imagers. We acknowledge privacy concerns in processing images—but the imagers we envisioned will not be functioning as a typical camera where images are taken for record or pleasure, but as a visual sensor whose sole function is to sense activities; as a result, any images captured will only be processed locally on device.

It is our belief that imagers are the best place to focus our research energy in order to achieve significant progress in HAR—towards recognizing the full spectrum of human activities. Our findings demonstrate that imagers offer a far superior source of information than accelerometer sensors for activity recognition, and currently the algorithmic and hardware advancements are in place to make their adoption feasible. If we place our research focus on imagers, we anticipate further enhancement that could lead to their widespread deployment and a vast array of sensing opportunities in HAR. We hope our work will lay the foundation for subsequent research exploring image-based HAR.

2 HOW FAR HAVE WE COME?

We begin our case for imager-based activity recognition in place of acceleration-based sensing by considering where our attention to accelerometers has got us in activity recognition. In the remaining discussion, we refer to the problem of HAR exclusively as the recognition of activities from sensor data through the use of machine learning models. We review accelerometer-based HAR, summarise current challenges and finally examine how progress in HAR has stalled as a result.

A Brief History. Accelerometers respond to the intensity and frequency of motions, allowing them to provide motion-based features especially useful for characterising repetitive motions with a short time window [5]. Activities such as sitting, standing, walking can be recognized efficiently using accelerometer data. The use of accelerometer data to recognize human activities have spanned almost two decades, with many initial works focused on recognising ambulation and posture [25]. In 2004, Bao and Intille used multiple accelerometers to recognise 20 activities in a naturalistic setting, extending to daily activities such as watching TV and folding laundry [4]. Since then, numerous accelerometer-based activity recognition works have followed [8], and accelerometers are perhaps the most frequently used body-worn sensor in HAR. The ubiquity of accelerometer-based activity recognition works is further cemented by the inclusion of accelerometers in almost all publicly-available datasets [29, 30, 38], which are used as benchmarks and thus define the scope of baseline activity recognition tasks. Beyond activity recognition, accelerometers have been used in applications including monitoring of physical activities, breathing, disease rehabilitation [27], driving pattern analysis [18] and skill assessment [21].

Common Struggles. With their motion-based features, accelerometers are most useful for identifying activities with characteristic motions, such as walking patterns. However, even within the regime of elementary actions, accelerometer-based sensing often struggles to overcome key challenges such as individual and environmental variations, sensor diversity and sensor placement issues, to which many common activities are prone [22]. In turn, additional resources, often in the form of collecting more data or developing more complex models, are required to overcome the confusion between activities or to provide person-specific training.

A fundamental challenge arises when accelerometers are used to recognise more complex activities without distinctive motions: Accelerometers inherently lack the dimension to caption any important contextual and social information beyond motion. Thus, the recognition of the full spectrum of human activities is unattainable with accelerometers.

Another widely recognised challenge with accelerometers is the difficulty of labelling the data collected, which prohibits the development of large datasets useful for deep learning techniques. Unless there is a clear ground truth was obtained closed to data collection (by recollection or additional visual data source), it is nearly impossible to label the data.

Stalled Progress. We perform the following analysis to answer our core question: Have we really made progress with accelerometer-based activity recognition or has it stalled? We review current models and datasets, in particular, the 18-Class Gesture Recognition task from the OPPORTUNITY Activity Recognition Dataset [7]. We make three key observations which indicate stalled progress:

Accuracy is not improving. Though our learning models have matured, activity recognition has seen no dramatic jump in performance. Figure 1 shows the weighted F1-score reported for the Opportunity gesture recognition task following its publication. Despite an initial increase in modelling power brought by Deep Neural Networks (DNN), introducing increasingly complex models has only led to scattered performance improvements in the past 5 years.

Limited Activity Labels. The variety of activity labels present in publicly-available HAR datasets is limited and confined to low to medium-level activities lacking any fine-grained or contextual descriptions. The labelling of the activities has perhaps been restricted to activities that motion-based models have some hope of distinguishing, which means we can never expect activity classes at the level of differentiating eating candy versus taking pills. In most

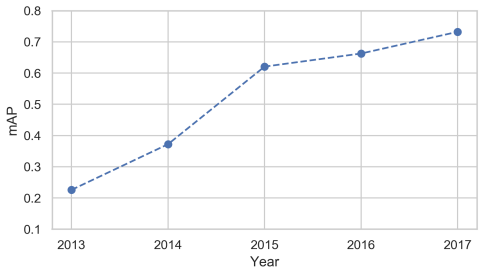


Figure 2: Best mAP achieved on ImageNet yearly between 2013–2017. Unlike Figure 1, we observe that performance for computer vision tasks—widely recognised as being challenging—have seen dramatic improvements in recent years thanks to deep learning.

datasets, a large percentage of activities belong to a 'null' or 'other' class which, prospectively, we have no chance of understanding what was actually performed from the acceleration traces.

Alarming confusions. Current inertial-based HAR models are far from robust and reliable. Lack-of-common-sense errors are commonplace, e.g. confusing 'drink from cup' with 'opening dishwasher' in Opportunity gesture recognition [37]. The performance of activity recognition measured by different metrics can also be drastically different—more recent publications consider sample-based measurements to be more relevant (shown in green in Figure 1) which only give the state-of-the-art F1-scores at low 70s [13], not to mention that these models already use many more sensors than are realistically deployable, as Opportunity collects data from accelerometers, gyroscopes, magnetometers at multiple body locations.

3 ARE IMAGERS THE ANSWER?

It appears we have reached the end of the road for HAR with accelerometers. We believe solutions exist in sensors that capture far richer observations about activities and context. We believe that *imagers*, in an embedded form which model images locally, are well-positioned to further HAR by enabling granular and contextual activity recognition. In what follows, we support our argument by pointing out that images are a rich source of information, and that the progress in computer vision provides a foundation from which we could build efficient, cheap and accurate imager-based HAR.

Not the first time. The idea that images are informative is not new, as supported by the vast number of activity recognition works which have used the visual modality in the form of wearable cameras, such as glasses [36], head [12], wrist [24], with notable wearable devices such as the SenseCam [16]. Activities recognizable using images include fine-grained activities (making tea vs coffee), social interactions [12] and context. [8] includes an extensive review of wearable camera-based activity recognition works. Other than wearable cameras, visual HAR based on stationary cameras has been extensively studied by the computer vision community and we refer readers to [39] for a survey in this area.

Information-Rich Images. Image data contains substantially more information than acceleration traces. Visual details captured in images may be high-level scene features quantifying the motion or contextualizing the scene, or they may be fine-grained details which specify an object that a person is interacting with [8]. The rich information provided by images not only enables more granular definitions of activity classes, it also provides abundant context for open-ended study of the subjects' behaviour.

We illustrate this richness in information by conducting a simple comparison between the data captured in the two forms: images and acceleration traces. We mounted two smartphones on a subject's dominant hand, each capturing one modality; we use the app *Accelerometer* [11] for recording acceleration, and the phone's video function for taking images. Figure 3 juxtaposes the images with the acceleration traces when two different activities were performed: typing on a phone and typing on a computer. The differences in the acceleration traces are hardly discernible nor intuitive as compared to that presented by the images. In addition, we can visually infer information such as objects, brands, environment, social setting simply from looking at these images; such additional information can be easily interrogated from the images through further processing and learning.

Deep Learning. More than ever, we have the image processing technology to be able to make use of the rich information in images, especially with the advances in Convolutional Neural Networks (CNNs). Figure 2 shows the yearly mean average precision (mAP) reported in the ImageNet visual recognition challenge [9], a benchmark in the computer visual community. There is an impressive and steady improvement in performance since the introduction of CNNs, and today the model performance has already surpassed that of human labelling; this is in stark contrast to the stalled progress seen in HAR. More than any other modality, the pairing of images and CNNs makes it practical to extract powerful information reliably enough to make the inference of complex human activities viable. In the realm of visual HAR with stationary cameras, state-of-the-art performance on the challenging (400-class) Kinetics dataset [19] has achieved top-1 accuracy exceeding 60%, although there is work to be done in leveraging these models for imager-based HAR, as will be discussed in Section 5.

Now is the time. Finally, it is possible to accelerate CNNs substantially—to the point that on-device inference is viable, which is precisely what is needed for the required combination of images, CNNs and tiny devices. We present a detailed feasibility analysis in Section 4.

Multi-sensor? We do not reject a multi-sensor scenario, especially if ample resources are available in terms of energy and computational costs. However, we view the multitude of common low-dimensional sensors (e.g. magnetometer, gyroscope) as having only a supplementary role supporting the rich information collected by imagers. Imagers remain the best choice amongst common inertial and non-inertial sensors (including microphones) due to the rich information they provide that can be used for disambiguating activities and avoiding confusions.

Privacy. We envision imagers to have only one function: to visually sense activities, and so any image captured would only be processed *locally* on the device. Running on-device recognition models prevents private visual information from ever leaving the local device end to the cloud. Concurrently, techniques developed

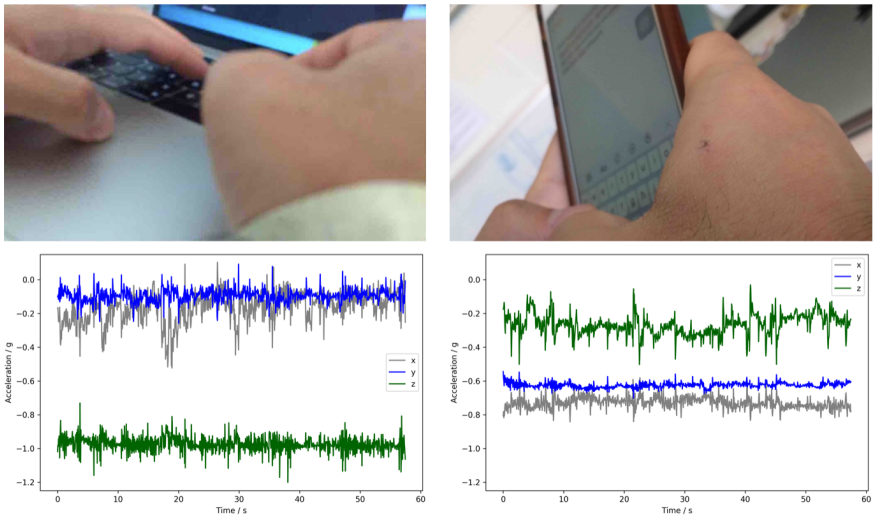


Figure 3: Image and accelerometer data collected from devices located at the wrist. Left: using a laptop, right: using a phone. The dominant difference in the acceleration data alignment is caused by differences in hand orientation, which can hardly be a robust feature for accurate predictions when variations within and between persons are considered. On the other hand, the differences between the two images are clear and comprehensible. A modern CNN would likely be able to disambiguate between these two images classes.

in extreme-low-resolution activity recognition [31] may be incorporated to achieve privacy by using low-resolution hardware (which further reduces computational costs).

While prior works using image-based approaches for activity recognition exist, they largely came before CNNs, as well as other efficient on-device technologies—two key pieces to the viability of image-based HAR which were previously missing, and without which it was not worth pushing for mainstream adoption of imagers for HAR. *Now* is the ideal time to move away from accelerometers and adopt imagers for activity recognition, due to the maturity of these complementary components. As a community, we now have the techniques to build models that allow us to understand images at a higher level than ever before, and the ability to run these models on smaller and more efficient devices than those envisaged even 5 years ago.

4 IS IT PRACTICAL TO USE IMAGERS?

Conventional wisdom suggests an image-based wearable is impractical. In this section, we show that this belief has been outdated by advances in image sensing, microprocessors and machine learning.

Image Sensing Technology. The current power consumption associated with collecting images is sufficiently low that it does not represent a bottleneck. Over the past decade, both academia and industry have contributed to a substantial reduction in the power consumption of image sensors, and it is on the trajectory towards increasing efficiency by another order of magnitude [15].

Low form-factor imagers can now be purchased easily off-the-shelf. One such imager, capable of capturing grayscale QQVGA images at 30 frames per second (FPS), can do so using less than 1mW [1]. We can expect a lower frame rate for activity recognition as there is no need to capture images at a frequency as high as 30FPS, hence power consumption can be reduced even further.

Quantity	Value
Latency of MobileNet ($\alpha = 0.25$, 160×160 input)	165ms
Microcontroller and camera run power	170mW
Microcontroller sleep power	2mW
Battery capacity	0.56Ah
Safety factor	0.7
Runtime	**13 hours**

Table 1: Summary of battery life calculation using an STM32H7 microcontroller and a 3.7V 150mAh battery, which represent realistic hardware choices for a wearable device.

On-Device Image Recognition. On-device modelling is vital to imager-based HAR: not only is it more efficient, it is also a solution to privacy concerns. A number of techniques have been proposed recently to allow neural networks to run on resource-constrained platforms such as wearable devices [14, 23, 32]. With techniques such as quantization and depthwise-separable convolutions it is now possible to run CNNs such as MobileNet [17] which obtain acceptable accuracy on images from ImageNet on ARM Cortex-M hardware, with sufficiently low latency for real-time use. We also expect the inference performance on low-power microcontrollers to improve rapidly: specialised RISC-V devices targeting this market are already available [2], and ARM have announced several features that will vastly improve their next generation of microcontrollers in this area [3].

Estimating Power Consumption. We provide an estimate of the expected battery life using existing hardware in Table 1. In our calculations, we consider the cost of deploying a MobileNet

model onto a high-performance microcontroller (STM32H7), which proceed as follows: According to [6], it takes 165ms to run a MobileNet with a width multiplier 0.25 and input shape 160×160×3. The microcontroller consumes ~170mW when running at maximum frequency, and under 2mW during sleep. This comes to an average power consumption of ~30mW if processing at 1 fps. We also assume a realistic battery for a wearable device would be a 150mAh 3.7V LiPo. With a safety factor of 0.7 to account for further losses, we obtain a battery life of *13 hours of continuous monitoring*, including the cost of capturing images.

Since the considered microcontroller is not designated as a low-power model, the power consumption may be assumed to reduce further using lower power manufacturing nodes. Also, the power consumption is dominated by the inference latency, which is likely to decrease over time with the integration of special-purpose accelerators or new instruction set extensions. However, even without these factors, we still obtain a useful battery lifetime, which could be extended by duty cycling readings where appropriate.

Image Acquisition Many statistical and learning-based methods exist to overcome low-light level and blurring issues. Using other imaging modalities (e.g. IR, depth) could also alleviate these. One may also install multiple imagers at different viewpoints, though additional strains on resources are expected due to increases in model size, memory pressure and inference time. A simple concatenation pipeline for fusing images with CNNs would mean inference time scaling linearly with the number of cameras used.

Form Factor. We believe imagers would be best placed in a form factor that is wearable and socially-acceptable while allowing for unobstructed capture of images. The choice of form factor is closely related to the orientation of imagers, the availability of continuous image streams as well as lighting conditions. Glasses are a good candidate as they can provide an unobstructed view of the wearer's current focus, which might be difficult in other cases for which specific design considerations are needed. The smartwatch is a popular form of body-worn device that imagers could be integrated into; here, the major challenge would be occlusion by clothing; this issue can be mitigated by carefully designing the device so that the sensors are placed as far down the arm as possible, along with using fish-eye lenses to increase field of view; Figure 4 shows an artist's conception of one such device, which is one of many potential form factors for imager-based HAR.

Not All Scenarios. While our discussion of imagers for activity recognition has mainly existed in the context of the core HAR task of identifying activities, we acknowledge that there are certain domains adjacent to activity recognition for which imagers might be harder to use or perform worse than accelerometers. In particular, imagers may fall short for sensing applications looking to measure the physical level of activities (e.g. detecting tremble levels in freezing of gait episodes of Parkinson's patients [26]), or to assess the quality of activities (e.g. skill level [21]).

5 THE ROAD AHEAD

In this section, we present a research agenda which will make imager-based activity recognition viable.

Modelling. We discuss key issues in modelling imagers.

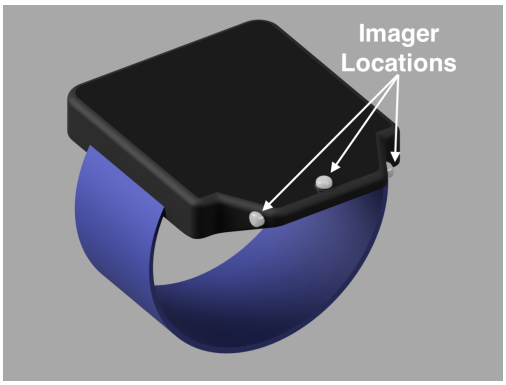


Figure 4: Conceptual image of a wrist-worn device with multiple embedded image sensors facing different directions. This device incorporates three sensors in a small protrusion along one side of the device in order to alleviate the problem of occlusion by clothing. This device is one of many potential form factors for imager-based HAR, which in this case takes a similar style to the common smartwatch.

From images to activities. Imager-based HAR methodologies are not extensively studied, though there are many shared similar challenges with existing fields, such as egocentric image processing, odometry and object recognition. A key problem is to effectively model sparse snapshots of activities which also have a temporal relationship. An approach, given the sophistication of visual object recognition, is a multi-step approach: recognize objects then activities [28]. Much can also be learnt from visual action recognition methods (e.g. [34]), although activities of interest in HAR might differ, and these models typically consume high-frame-rate videos not likely to be available from imager-embedded devices.

Training. A core challenge is the lack of imager-generated datasets annotated for activities, especially from various ego-centric viewpoints from the body (e.g. wrist, belt, foot). Large image datasets used in object recognition (e.g. ImageNet [9]) or in visual action recognition (e.g. Kinetics [19]) present potential opportunities for transfer learning from these respective domains to that of naturalistic egcocentric images; such transfer learning schemes will be highly fruitful by leveraging non-sensitive, standardized datasets.

Multi-views. To effectively combine data generated from multiple imagers embedded on a device, one might leverage multi-camera fusion from robotics [20], though accomplishing this from an egocentric perspective with sporadic actions is a challenge.

Community efforts. The availability of accessible imager datasets is vital to facilitating modelling efforts in imager-based HAR. The ideal scenario is the development of such a dataset, with the community also defining a wide scope of activity labels, so that models can be built to recognize activities of different complexity at different stages of development. Doing so while addressing privacy concerns of collecting such datasets would be another open challenge.

Image Sensors. Our feasibility analysis had been run with off-the-shelf image sensors sub-optimal for imager-based HAR. The camera selected also provides better specifications than necessary: both the frame rate and image size were larger than what current hardware

can process on-device. It will be possible to build cameras that more closely match the target application, with lower power consumption and manufacturing costs, e.g. adopting voltage reduction [15].

Hardware Design. Finally, machine learning on low-power devices is an area that is receiving significant attention from both the academic and industrial community. Microcontrollers incorporating extensions or accelerators suited for machine learning workloads are already becoming a reality [2, 3], which would cause a significant reduction in inference latency. There is concurrent work on how to train networks that utilise this hardware optimally.

6 CONCLUSION

Accelerometers are a dead-end for activity recognition: they do not offer enough information and our reliance on them has led to a stalled progress in HAR. In order to recognize the full spectrum of human activities we should adopt imagers as the default HAR sensor, since it is now possible to exploit the information-richness of images with the rise of energy-efficient CNNs. Collectively, our study argues that now is the time to pursue HAR using imagers, and calls for further exploration into overcoming the existing challenges for imager-based HAR.

ACKNOWLEDGEMENT

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) under Grant No.: DTP (EP/R513295/1) and MOA (EP/S001530/), and Samsung AI. We would like to thank Dr. Petteri Nurmi and other anonymous reviewers for their feedback throughout the submission process for ACM HotMobile 2020, and William Ip for facilitating the experiments.

REFERENCES

[1] HM01B0 « Himax Technologies, Inc. https://www.himax.com.tw.
[2] Kendryte - Kendryte. https://kendryte.com/.
[3] Arm Enables Custom Instructions for Embedded CPUs. https://www.arm.com/company/news/2019/10/arm-enables-custom-instructions-for-embedded-cpus.
[4] L. Bao and S. S. Intille. 2004. Activity Recognition from User-Annotated Acceleration Data. In *Pervasive*.
[5] A. Bulling et al. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Computing Surveys (CSUR)* 46, 3 (2014), 1–33.
[6] A. Capotondi. 2019. EEESlab/Mobilenet_v1_stm32_cmsis_nn. Energy-Efficient Embedded Systems Laboratory.
[7] R. Chavarriaga et al. The Opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recognition Letters* (2013).
[8] M. Cornacchia et al. A survey on activity detection and classification using wearable sensors. *IEEE Sensors Journal* (2016)
[9] J. Deng et al. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*
[10] A. R. Doherty et al. Using wearable cameras to categorise type and context of accelerometer-identified episodes of physical activity. *International Journal of Behavioral Nutrition and Physical Activity* (2013)
[11] DreamArc. 2019. Accelerometer App.
[12] A. Fathi et al. Social interactions: A first-person perspective. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*
[13] Yu Guan et al. Ensembles of deep LSTM learners for activity recognition using wearables. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2017)
[14] S. Han et al. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
[15] S. Hanson et al. A 0.5 V Sub-Microwatt CMOS Image Sensor With Pulse-Width Modulation Read-Out. *2010 IEEE JSSC*
[16] S. Hodges et al. 2006. SenseCam: A retrospective memory aid. In *International Conference on Ubiquitous Computing*. Springer.
[17] A. G. Howard et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* (April 2017).
[18] D. A. Johnson and Mohan M Trivedi. 2011. Driving style recognition using a smartphone as a sensor platform. In *2011 IEEE ITSC*, 1609–1615.
[19] W. Kay et al. 2017. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950* (2017).
[20] B. Keyes et al. Camera placement and multi-camera fusion for remote robot operation. In *Proceedings of the IEEE SSSR (2006)*.
[21] A. Khan et al. 2015. Beyond activity recognition: skill assessment from accelerometer data. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 1155–1166.
[22] N. D. Lane. 2011. *Community-Guided Mobile Phone Sensing Systems*. Ph.D. Dissertation. Dartmouth College.
[23] N. D. Lane et al. Squeezing deep learning into mobile and embedded devices. *IEEE Pervasive Computing* 16, 3 (2017), 82–88.
[24] T. Maekawa et al. Object-Based Activity Recognition with Heterogeneous Sensors on Wrist. In *Pervasive Computing* (2010)
[25] J. Mantyjarvi et al. Recognizing human motion with multiple acceleration sensors. In *IEEE International Conference on Systems, Man and Cybernetics* (2001)
[26] K. Niazmand et al. Freezing of Gait detection in Parkinson's disease using accelerometer based smart clothes. In *2011 IEEE Biomedical Circuits and Systems Conference (BioCAS)*.
[27] S. Patel et al. A review of wearable sensors and systems with application in rehabilitation. *Journal of neuroengineering and rehabilitation* 9, 1 (2012), 21.
[28] M. Philipose et al. Inferring activities from interactions with objects. *IEEE Pervasive Computing* 3, 4 (2004), 50–57.
[29] A. Reiss and D. Stricker. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th International Symposium on Wearable Computers*
[30] D. Roggen et al. Collecting complex activity datasets in highly rich networked sensor environments. In *2010 Seventh international conference on networked sensing systems (INSS)*
[31] M. S. Ryoo et al. 2018. Extreme low resolution activity recognition with multi-siamese embedding learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
[32] V. Sze et al. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.
[33] J. Wang et al. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters* (2019)
[34] L. Wang et al. 2015. Action recognition with trajectory-pooled deep-convolutional descriptors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.
[35] D. Weinland et al. A survey of vision-based methods for action representation, segmentation and recognition. *Computer Vision and Image Understanding* 115, 2 (2011), 224 – 241.
[36] J. Windau and L. Itti. Situation awareness via sensor-equipped eyeglasses. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*
[37] J. Yang et al. 2015. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.
[38] P. Zappi et al. 2008. Activity recognition from on-body sensors: accuracy-power trade-off by dynamic sensor selection. In *European Conference on Wireless Sensor Networks*
[39] H. Zhang et al. 2019. A Comprehensive Survey of Vision-Based Human Action Recognition Methods. *Sensors* 19, 5 (2019), 1005.

DEGREE-QUANT: QUANTIZATION-AWARE TRAINING FOR GRAPH NEURAL NETWORKS

Shyam A. Tailor*
Computer Laboratory
University of Cambridge

Javier Fernandez-Marques*
Dept. of Computer Science
University of Oxford

Nicholas D. Lane
University of Cambridge
& Samsung AI Center

ABSTRACT

Graph neural networks (GNNs) have demonstrated strong performance on a wide variety of tasks due to their ability to model non-uniform structured data. Despite their promise, there exists little research exploring methods to make them more efficient at inference time. In this work, we explore the viability of training quantized GNNs, enabling the usage of low precision integer arithmetic during inference. We identify the sources of error that uniquely arise when attempting to quantize GNNs, and propose an architecturally-agnostic method, *Degree-Quant*, to improve performance over existing quantization-aware training baselines commonly used on other architectures, such as CNNs. We validate our method on six datasets and show, unlike previous attempts, that models generalize to unseen graphs. Models trained with Degree-Quant for INT8 quantization perform as well as FP32 models in most cases; for INT4 models, we obtain up to 26% gains over the baselines. Our work enables up to 4.7× speedups on CPU when using INT8 arithmetic.

1 INTRODUCTION

Graph neural networks (GNNs) have received substantial attention in recent years due to their ability to model irregularly structured data. As a result, they are extensively used for applications as diverse as molecular interactions (Duvenaud et al., 2015; Wu et al., 2017), social networks (Hamilton et al., 2017), recommendation systems (van den Berg et al., 2017) or program understanding (Allamanis et al., 2018). Recent advancements have centered around building more sophisticated models including new types of layers (Kipf & Welling, 2017; Velickovic et al., 2018; Xu et al., 2019) and better aggregation functions (Corso et al., 2020). However, despite GNNs having few model parameters, the compute required for each application remains tightly coupled to the input graph size. A 2-layer GCN model with 32 hidden units would result in a model size of just 81KB but requires 19 GigaOPs to process the entire Reddit graph. We illustrate this growth in fig. 1.

One major challenge with graph architectures is therefore performing inference efficiently, which limits the applications they can be deployed for. For example, GNNs have been combined with CNNs for SLAM feature matching (Sarlin et al., 2019), however it is not possible to deploy this technique on smartphones, or even smaller devices, whose neural network accelerators often do not implement floating point arithmetic, and instead favour more efficient integer arithmetic. Integer quantization is one way to lower the compute and memory budget required to perform inference, without necessarily requiring modifications to the model architecture; this is also useful for model serving in data centers.

Although quantization has been well studied for CNNs and language models (Jacob et al., 2017; Wang et al., 2018; Zafrir et al., 2019; Prato et al., 2019), there remains little work addressing GNN efficiency (Mukkara et al., 2018; Jia et al., 2020). To the best of our knowledge, there is no work explicitly characterising the issues that arise when quantizing GNNs or showing latency benefits of using low-precision arithmetic in common applications for GNNs. The recent work of Wang et al. (2020) explores only binarized embeddings of a single graph type (citation networks). In Feng et al. (2020) a heterogeneous quantization framework assigns different bits to embedding and attention coefficients in each layer while maintaining the weights at full precision (FP32). Due to the mismatch in operands' bit-width the majority of the operations are performed at FP32 after data casting, making it impractical to use in general purpose hardware such as CPUs or GPUs. In addition they do not demonstrate how to train networks which generalize to *unseen* input graphs. Our framework relies

*Equal contribution. Correspondence to: Shyam Tailor <sat62@cam.ac.uk>

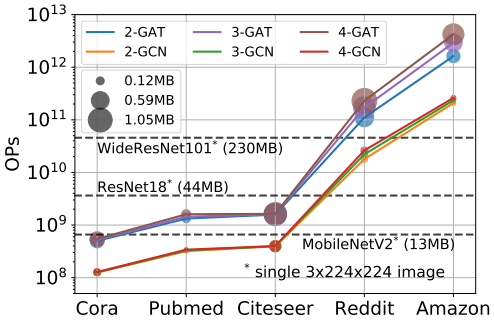


Figure 1: Despite GNN model sizes rarely exceeding 1MB, the OPs needed for inference grows at least linearly with the size of the dataset and node features. GNNs with models sizes $100\times$ smaller than popular CNNs require many more OPs to process large graphs.

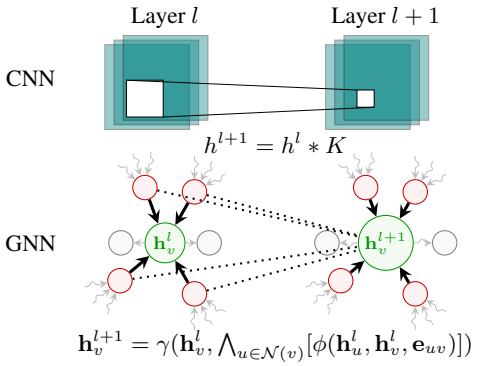


Figure 2: While CNNs operate on regular grids, GNNs operate on graphs with varying topology. A node's neighborhood size and ordering varies for GNNs. Both architectures use weight sharing.

upon uniform quantization applied to all elements in the network and uses bit-widths (8-bit and 4-bit) that are supported by off-the-shelf hardware such as CPUs and GPU for which efficient low-level operators for common operations found in GNNs exists.

Architecturally, CNNs and GNNs have several similarities, as shown in fig. 2. In a single graph, the variance in node degree can be substantial, which may cause the activations at GNN nodes to have a wider variance than would be observed at different spatial coordinates in a CNN architecture, which has a fixed effective node degree. This irregularity makes quantization less straightforward for GNNs.

This work considers the motivations and problems associated with quantization of graph architectures, and provides the following contributions:

- The explanation of the sources of degradation in GNNs when using lower precision arithmetic. We show how the choice of straight-through estimator (STE) implementation, node degree, and method for tracking quantization statistics significantly impacts performance.
- An *architecture-agnostic* method for quantization-aware training on graphs, *Degree-Quant* (DQ), which results in INT8 models often performing as well as their FP32 counterparts. At INT4, models trained with DQ typically outperform quantized baselines by over 20%. We show, unlike previous work, that models trained with DQ generalize to *unseen graphs*.
- We show that quantized networks achieve up to $4.7\times$ speedups on CPU with INT8 arithmetic, relative to full precision floating point, with 4-$8\times$ reductions in runtime memory usage.

2 Background

2.1 Message Passing Neural Networks (MPNNs)

Many popular GNN architectures may be viewed as generalizations of CNN architectures to an irregular domain: at a high level, graph architectures attempt to build representations based on a node's neighborhood. Unlike CNNs, however, this neighborhood does not have a fixed ordering or size. This work considers GNN architectures conforming to the MPNN paradigm (Gilmer et al., 2017). A graph $\mathcal{G} = (V, E)$ has node features $\mathbf{X} \in \mathbb{R}^{N \times F}$, an incidence matrix $\mathbf{I} \in \mathbb{N}^{2 \times E}$, and optionally D-dimensional edge features $\mathbf{E} \in \mathbb{R}^{E \times D}$. The forward pass through an MPNN layer consists of message passing, aggregation and update phases: $\mathbf{h}_{l+1}^{(i)} = \gamma(\mathbf{h}_l^{(i)}, \bigwedge_{j \in \mathcal{N}(i)}[\phi(\mathbf{h}_l^{(j)}, \mathbf{h}_l^{(i)}, \mathbf{e}_{ij})])$. Messages from node u to node v are calculated using function ϕ, and are aggregated using a permutation-invariant function $\bigwedge$. The features at v are subsequently updated using γ.

We focus on three architectures with corresponding update rules:

1. Graph Convolution Network (GCN): $\mathbf{h}_{l+1}^{(i)} = \sum_{j \in \mathcal{N}(i) \cup \{i\}} (\frac{1}{\sqrt{d_i d_j}} \mathbf{W} \mathbf{h}_l^{(j)})$ (Kipf & Welling, 2017), where d_i refers to the degree of node i.

2. Graph Attention Network (GAT): $\mathbf{h}_{l+1}^{(i)} = \alpha_{i,i}\mathbf{W}\mathbf{h}_l^{(i)} + \sum_{j\in\mathcal{N}(i)}(\alpha_{i,j}\mathbf{W}\mathbf{h}_l^{(j)})$, where α represent attention coefficients (Velickovic et al., 2018).
3. Graph Isomorphism Network (GIN): $\mathbf{h}_{l+1}^{(i)} = f_{\mathbf{\Theta}}[(1+\epsilon)\mathbf{h}_l^{(i)} + \sum_{j\in\mathcal{N}(i)}\mathbf{h}_l^{(j)}]$, where f is a learnable function (e.g. a MLP) and ϵ is a learnable constant (Xu et al., 2019).

2.2 Quantization for Non-Graph Neural Networks

Quantization allows for model size reduction and inference speedup without changing the model architecture. While there exists extensive studies of the impact of quantization at different bit-widths (Courbariaux et al., 2015; Han et al., 2015; Louizos et al., 2017) and data formats (Micikevicius et al., 2017; Carmichael et al., 2018; Kalamkar et al., 2019), it is 8-bit integer (INT8) quantization that has attracted the most attention. This is due to INT8 models reaching comparable accuracy levels to FP32 models (Krishnamoorthi, 2018; Jacob et al., 2017), offer a $4\times$ model compression, and result in inference speedups on off-the-shelf hardware as 8-bit arithmetic is widely supported.

Quantization-aware training (QAT) has become the *de facto* approach towards designing robust quantized models with low error (Wang et al., 2018; Zafrir et al., 2019; Wang et al., 2018). In their simplest forms, QAT schemes involve exposing the numerical errors introduced by quantization by simulating it on the forward pass Jacob et al. (2017) and make use of a straight-through estimator (STE) (Bengio et al., 2013) to compute the gradients—as if no quantization had been applied.

To reach performance comparable to FP32 models, QAT schemes often rely on other techniques such as *gradient clipping*, to mask gradient updates based on the largest representable value at a given bit-width; noisy QAT, which stochastically applies QAT to a portion of the weights at each training step (Fan et al., 2020); or the re-ordering of layers (Sheng et al., 2018; Alizadeh et al., 2019).

3 Quantization for GNNs

In this section, we build an intuition for why GNNs would fail with low precision arithmetic by identifying the sources of error that will disproportionately affect the accuracy of a low precision model. Using this insight, we propose our technique for QAT with GNNs, *Degree-Quant*. Our analysis focuses on three models: GCN, GAT and GIN. This choice was made as we believe that these are among the most popular graph architectures, with strong performance on a variety of tasks (Dwivedi et al., 2020), while also being representative of different trends in the literature.

3.1 Sources of Error

QAT relies upon the STE to make an estimate of the gradient despite the non-differentiable rounding operation in the forward pass. If this approximation is inaccurate, however, then poor performance will be obtained. In GNN layers, we identify the aggregation phase, where nodes combine messages from a varying number of neighbors in a permutation-invariant fashion, as a source of substantial numerical error, especially at nodes with high in-degree. Outputs from aggregation have magnitudes that vary significantly depending on a node's in-degree: as it increases, the variance of aggregation values will increase.[1] Over the course of training $q_{\min}$ and $q_{\max}$, the quantization range statistics, become severely distorted by infrequent outliers, reducing the resolution for the vast majority of values observed. This reults in increased rounding error for nodes with smaller in-degrees. Controlling $q_{\min}$ and $q_{\max}$ hence becomes a trade-off balancing *truncation error* and *rounding error*.

We can derive how the mean and variance of the aggregation output values vary as node in-degree, n, increases for each of the three GNN layers. Suppose we model incoming message values for a single output dimension with random variables X_i, without making assumptions on their exact distribution or independence. Further, we use Y_n as the random variable representing the value of node output after the aggregation step. With GIN layers, we have $Y_n = (1+\epsilon)X_0 + \sum_{i=1}^{n} X_i$. It is trivial to prove that $\mathbb{E}(Y_n) = \mathcal{O}(n)$. The variance of the aggregation output is also $\mathcal{O}(n)$ in the case that that $\sum_{i\neq j}\text{Cov}(X_i, X_j) \ll \sum_i \text{Var}(X_i)$. We note that if $\sum_{i\neq j}\text{Cov}(X_i, X_j)$ is large then it implies that the network has learned highly redundant features, and may be a sign of over-fitting. Similar

[1] The reader should note that we are not referring to the concept of estimator variance, which is the subject of sampling based approaches—we are exclusively discussing the variance of values immediately after aggregation.

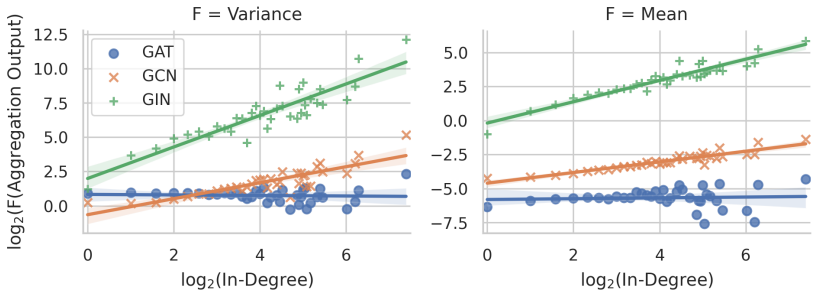


Figure 3: Analysis of values collected immediately after aggregation at the final layer of FP32 GNNs trained on Cora. Generated using channel data collected from 100 runs for each architecture. As in-degree grows, so does the mean and variance of channel values after aggregation.

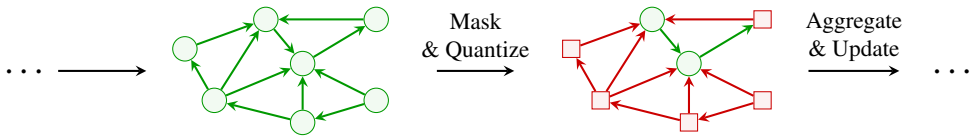


Figure 4: High-level view of the stochastic element of Degree-Quant. Masked (high in-degree) nodes, in green, operate at full precision, while unmasked nodes (red) operate at reduced precision. High in-degree nodes contribute most to poor gradient estimates, hence they are stochastically masked more often.

arguments can be made for GCN and GAT layers; we would expect GCN aggregation values to grow like $\mathcal{O}(\sqrt{n})$, and GAT aggregation values to remain constant ($\mathcal{O}(1)$) due to the attention coefficients.

We empirically validate these predictions on networks trained on the Cora dataset; results are plotted in fig. 3. We see from the log-log plot that the aggregation values do follow the trends predicted, and that for the values of in-degree in the plot (up to 168) the covariance terms can be neglected. As expected, the variance and mean of the aggregated output grow fastest for GIN, and are roughly constant for GAT as in-degree increases. From this empirical evidence, it would be expected that GIN layers are most affected by quantization.

By using GIN and GCN as examples, we can see how aggregation error causes error in weight updates. Suppose we consider a GIN layer incorporating one weight matrix in the update function i.e. $\mathbf{h}_{l+1}^{(i)} = f(\mathbf{W}\mathbf{y}_{\text{GIN}}^{(i)})$, where f is an activation function, $\mathbf{y}_{\text{GIN}}^{(i)} = (1+\epsilon)\mathbf{h}_l^{(i)} + \sum_{j\in\mathcal{N}(i)} \mathbf{h}_l^{(j)}$, and $\mathcal{N}(i)$ denotes the in-neighbors of node i. The derivative of the loss with respect to the weights is:

GIN

$$\frac{\partial\mathcal{L}}{\partial\mathbf{W}} = \sum_{i=1}^{|V|}\left(\frac{\partial\mathcal{L}}{\partial\mathbf{h}_{l+1}^{(i)}} \circ f'(\mathbf{W}\mathbf{y}_{\text{GIN}}^{(i)})\right)\mathbf{y}_{\text{GIN}}^{(i)\top}$$

GCN

$$\frac{\partial\mathcal{L}}{\partial\mathbf{W}} = \sum_{i=1}^{|V|}\sum_{j\in\mathcal{N}(i)}\frac{1}{\sqrt{d_i d_j}}\left(\frac{\partial\mathcal{L}}{\partial\mathbf{h}_{l+1}^{(i)}} \circ f'(\mathbf{y}_{\text{GCN}}^{(i)})\right)\mathbf{h}_l^{(j)\top}$$

Where $\mathbf{y}_{\text{GCN}}^{(i)} = \sum_{k\in\mathcal{N}(i)}(\frac{1}{\sqrt{d_i d_k}}\mathbf{W}\mathbf{h}_l^{(j)})$. The larger the error in $\mathbf{y}_{\text{GIN}}^{(i)}$—caused by aggregation error—the greater the error in the weight gradients for GIN, which results in poorly performing models being obtained. The same argument applies to GCN, with both the $\mathbf{h}_l^{(j)\top}$ and $\mathbf{y}_{\text{GCN}}^{(i)}$ terms introducing aggregation error into the weight updates.

3.2 Our Method: Degree-Quant

To address these sources of error we propose *Degree-Quant* (DQ), a method for QAT with GNNs. We consider both *inaccurate weight updates* and *unrepresentative quantization ranges*.

Stochastic Masking to Improve Weight Update Accuracy. DQ aims to encourage more accurate weight updates by stochastically masking nodes in the network, and performing the forward pass at full precision. At each layer a binary node mask is generated; all masked nodes have the phases of the message passing, aggregation and update performed at full precision. This includes messages sent by masked nodes to other nodes, as shown in fig. 4. It is also important to note that the weights used at all nodes are the same quantized weights; this is motivated by the fact that our method is used

Algorithm 1 Degree-Quant (DQ). Functions accepting mask parameter $\mathbf{m}$ perform only the masked computations at full precision: intermediate tensors are *not* quantized. At test time masking is disabled.

procedure TRAINFORWARDPASS($\mathcal{G}, \mathbf{p}$)
  ▷ Calculate mask and *quantized* weights, Θ', which all operations share
  $\mathbf{m} \leftarrow$ BERNOULLI($\mathbf{p}$)
  $\Theta' \leftarrow$ QUANTIZE(Θ)
  ▷ Messages with masked sources are at full precision (excluding weights)
  $\mathcal{M} \leftarrow$MESSAGECALCULATE($\mathcal{G}, \Theta', \mathbf{m}$)
  $X \leftarrow$ QUANTIZE(AGGREGATE($\mathcal{M}, \Theta', \mathbf{m}$), $\mathbf{m}$) ▷ No quantization for masked nodes
  return UPDATE($X, \Theta', \mathbf{m}$) ▷ Quantized weights always used
end procedure

to encourage more accurate gradients to flow back to the weights through high in-degree nodes. At test time masking is disabled: all nodes operate at low precision.

To generate the mask, we pre-process each graph before training and create a vector of probabilities $\mathbf{p}$ with length equal to the number of nodes. At training time, mask $\mathbf{m}$ is generated by sampling using the Bernoulli distribution: $\mathbf{m} \sim \text{Bernoulli}(\mathbf{p})$. In our scheme p_i is higher if the in-degree of node i is large, as we find empirically that high in-degree nodes contribute most towards error in weight updates. We use a scheme with two hyperparameters, $p_{\min}$ and $p_{\max}$; nodes with the maximum in-degree are assigned $p_{\max}$ as their masking probability, with all other nodes assigned a probability calculated by interpolating between $p_{\min}$ and $p_{\max}$ based on their in-degree ranking in the graph.

Percentile Tracking of Quantization Ranges. Figure 3 demonstrates large fluctuations in the variance of the aggregation output as in-degree increases. Since these can disproportionately affect the ranges found by using min-max or momentum-based quantization, we propose using *percentiles*. While percentiles have been used for post-training quantization (Wu et al., 2020), we are the first (to the best of our knowledge) to propose making it a core part of QAT; we find it to be a key contributor to achieving consistent results with graphs. We are more aggressive than existing literature on the quantity we discard: we clip the top and bottom 0.1%, rather than 0.01%, as we observe the fluctuations to be a larger issue with GNNs than with CNNs or DNNs. Quantization ranges are more representative of the vast majority of values in this scheme, resulting in less *rounding error*.

We emphasize that a core contribution of DQ is that it is *architecture-agnostic*. Our method enables a wide variety of architectures to use low precision arithmetic at inference time. Our method is also *orthogonal*—and complementary—to other techniques for decreasing GNN computation requirements, such as sampling based methods which are used to reduce memory consumption (Zeng et al., 2020), or weight pruning (Blalock et al., 2020) approaches to achieve further model compression.

4 EXPERIMENTS

In this section we first analyse how the choice of quantization implementation affects performance of GNNs. We subsequently evaluate Degree-Quant against the strong baselines of: FP32, INT8-QAT and, INT8-QAT with stochastic masking of weights (Fan et al., 2020). We refer to this last approach as *noisy* QAT or nQAT. To make explicit that we are quantizing both weights and activations, we use the notation W8A8. We repeat the experiments at INT4. Our study evaluates performance on six datasets and includes both node-level and graph-level tasks. The datasets used were Cora, CiteSeer, ZINC, MNIST and CIFAR10 superpixels, and REDDIT-BINARY. Across all datasets INT8 models trained with Degree-Quant manage to recover most of the accuracy lost as a result of quantization. In some instances, DQ-INT8 outperform the extensively tuned FP32 baselines. For INT4, DQ outperforms all QAT baselines and results in double digits improvements over QAT-INT4 in some settings. Details about each dataset and our experimental setup can be found in appendix A.1.

4.1 IMPACT OF QUANTIZATION GRADIENT ESTIMATOR ON CONVERGENCE

The STE is a workaround for when the forward pass contains non-differentiable operations (e.g. rounding in QAT) that has been widely adopted in practice. While the choice of STE implementation generally results in marginal differences for CNNs—even for binary networks (Alizadeh et al., 2019)—it is unclear whether only marginal differences will also be observed for GNNs. Motivated

Dataset	Model Arch.	*vanilla* STE				STE with Gradient Clipping			
		min/max		momentum		min/max		momentum	
		W8A8	W4A4	W8A8	W4A4	W8A8	W4A4	W8A8	W4A4
Cora (Acc. %) ↑	GCN	$\mathbf{81.0 \pm 0.7}$	65.3 ± 4.9	42.3 ± 11.1	49.4 ± 8.8	80.8 ± 0.8	62.3 ± 5.2	66.9 ± 18.2	$\mathbf{77.2 \pm 2.5}$
	GAT	76.0 ± 2.2	16.8 ± 8.5	81.7 ± 1.3	$\mathbf{51.7 \pm 5.8}$	76.4 ± 2.6	15.4 ± 8.1	$\mathbf{81.9 \pm 0.7}$	47.4 ± 5.0
	GIN	69.9 ± 1.9	25.9 ± 2.6	49.2 ± 10.2	$\mathbf{42.8 \pm 4.0}$	69.2 ± 2.3	29.5 ± 3.5	$\mathbf{75.1 \pm 1.1}$	40.5 ± 5.0
MNIST (Acc. %) ↑	GCN	$\mathbf{90.4 \pm 0.2}$	51.3 ± 7.5	90.1 ± 0.5	$\mathbf{70.6 \pm 2.4}$	90.4 ± 0.3	54.8 ± 1.5	90.2 ± 0.4	10.3 ± 0.0
	GAT	$\mathbf{95.8 \pm 0.1}$	20.1 ± 3.3	95.7 ± 0.3	67.4 ± 3.2	95.7 ± 0.1	30.2 ± 7.4	95.7 ± 0.3	$\mathbf{76.3 \pm 1.2}$
	GIN	96.5 ± 0.3	62.4 ± 21.8	$\mathbf{96.7 \pm 0.2}$	$\mathbf{91.0 \pm 0.6}$	96.4 ± 0.4	19.5 ± 2.1	75.3 ± 18.1	10.8 ± 0.9
ZINC (Loss) ↓	GCN	0.486 ± 0.01	0.747 ± 0.02	0.509 ± 0.01	0.710 ± 0.05	0.495 ± 0.01	0.766 ± 0.02	$\mathbf{0.483 \pm 0.01}$	$\mathbf{0.692 \pm 0.01}$
	GAT	0.471 ± 0.01	0.740 ± 0.02	0.571 ± 0.03	$\mathbf{0.692 \pm 0.06}$	0.466 ± 0.01	0.759 ± 0.04	$\mathbf{0.463 \pm 0.01}$	0.717 ± 0.03
	GIN	0.393 ± 0.02	1.206 ± 0.27	$\mathbf{0.386 \pm 0.03}$	$\mathbf{0.572 \pm 0.02}$	0.390 ± 0.02	1.669 ± 0.10	0.388 ± 0.02	0.973 ± 0.24

Table 1: Impact on performance of four typical quantization implementations for INT8 and INT4. The configuration that resulted in best performing models for each dataset-model pair is bolded. Hyperparameters for each experiment were fine-tuned independently. **A major contribution of this work is identifying that seemingly unimportant choices in quantization implementation cause dramatic changes in performance.**

by this, we study the impact of four off-the-shelve quantization procedures on the three architectures evaluated for each type of dataset; the implementation details of each one is described in appendix A.3. We perform this experiment to ensure that we have the strongest possible QAT baselines. Results are shown in table 1. We found the choice quantization implementation to be highly dependent on the model architecture and type of problem to be solved: we see a much larger variance than is observed with CNNs; this is an important discovery for future work building on our study.

We observe a general trend in all INT4 experiments benefiting from momentum as it helps smoothing out the quantization statistics for the inherently noisy training stage at low bitwidths. This trend applies as well for the majority of INT8 experiments, while exhibiting little impact on MNIST. For INT8 Cora-GCN, large gradient norm values in the early stages of training (see fig. 5) mean that these models not benefit from momentum as quantization ranges fail to keep up with the rate of changes in tensor values; higher momentum can help but also leads to instability. In contrast, GAT has stable initial training dynamics, and hence obtains better results with momentum. For the molecules dataset ZINC, we consistently obtained lower regression loss when using momentum. We note that GIN models often suffer from higher performance degradation (as was first noted in fig. 3), specially at W4A4. This is not the case however for image datasets using superpixels. We believe that datasets with Gaussian-like node degree distributions (see fig. 9) are more tolerant of the imprecision introduced by quantization, compared to datasets with tailed distributions. We leave more in-depth analysis of how graph topology affects quantization as future work.

4.2 Obtaining Quantization baselines

Our FP32 results, which we obtain after extensive hyperparameter tuning, and those from the baselines are shown at the top of table 2. We observed large gains on MNIST, CIFAR10 and, ZINC.

For our QAT-INT8 and QAT-INT4 baselines, we use the quantization configurations informed by our analysis in section 4.1. For Citeseer we use the best resulting setup analysed for Cora, and for CIFAR-10 that from MNIST. Then, the hyperparameters for each experiment were fine tuned individually, including noise rate $n \in [0.5, 0.95]$ for nQAT experiments. QAT-INT8 and QAT-INT4 results in table 2 and QAT-INT4, with the exception of MNIST (an easy to classify dataset), corroborate our hypothesis that GIN layers are less resilient to quantization. This was first observed in fig. 3. In the case of ZINC, while all models results in noticeable degradation, GIN sees a more severe 16% increase of regression loss compared to our FP32 baseline. For QAT W4A4 an accuracy drop of over 35% and 47% is observed for Cora and Citeseer respectively. The stochasticity induced by nQAT helped in recovering some of the accuracy lost as a result of quantization for citation networks (both INT8 and INT4) but had little impact on other datasets and harmed performance in some cases.

4.3 Comparisons of Degree-Quant with Existing Quantization Approaches

Degree-Quant provides superior quantization for all GNN datasets and architectures. Our results with DQ are highlighted in gray in table 2 and table 3. Citation networks trained with DQ for W8A8 manage to recover most of the accuracy lost as a result of QAT and outperform most of nQAT baselines. In some instances DQ-W8A8 models outperform the reference FP32 baselines. At 4-bits,

Quant. Scheme	Model Arch.	Node Classification (Accuracy %) Cora ↑	Citeseer ↑	Graph Classification (Accuracy %) MNIST ↑	CIFAR-10 ↑	Graph Regression (Loss) ZINC ↓
Ref. (FP32)	GCN	81.4 ± 0.7	71.1 ± 0.7	90.0 ± 0.2	54.5 ± 0.1	0.469 ± 0.002
	GAT	83.1 ± 0.4	72.5 ± 0.7	95.6 ± 0.1	65.4 ± 0.4	0.463 ± 0.002
	GIN	77.6 ± 1.1	66.1 ± 0.9	93.9 ± 0.6	53.3 ± 3.7	0.414 ± 0.009
Ours (FP32)	GCN	81.2 ± 0.6	71.4 ± 0.9	90.9 ± 0.4	58.4 ± 0.5	0.450 ± 0.008
	GAT	83.2 ± 0.3	72.4 ± 0.8	95.8 ± 0.4	65.1 ± 0.8	0.455 ± 0.006
	GIN	77.9 ± 1.1	65.8 ± 1.5	96.4 ± 0.4	57.4 ± 0.7	0.334 ± 0.024
QAT (W8A8)	GCN	81.0 ± 0.7	71.3 ± 1.0	90.9 ± 0.2	56.4 ± 0.5	0.481 ± 0.029
	GAT	81.9 ± 0.7	71.2 ± 1.0	95.8 ± 0.3	66.3 ± 0.4	0.460 ± 0.005
	GIN	75.6 ± 1.2	63.0 ± 2.6	96.7 ± 0.2	52.4 ± 1.2	0.386 ± 0.025
nQAT (W8A8)	GCN	81.0 ± 0.8	70.7 ± 0.8	91.1 ± 0.1	56.2 ± 0.5	0.472 ± 0.015
	GAT	82.5 ± 0.5	71.2 ± 0.7	96.0 ± 0.1	66.7 ± 0.2	0.459 ± 0.007
	GIN	77.4 ± 1.3	65.1 ± 1.4	96.4 ± 0.3	52.7 ± 1.4	0.405 ± 0.016
DQ (W8A8)	GCN	81.7 ± 0.7 (**+0.7**)	71.0 ± 0.9 (**-0.3**)	90.9 ± 0.2 (**-0.2**)	56.3 ± 0.1 (**-0.1**)	0.434 ± 0.009 (**+9.8**)
	GAT	82.7 ± 0.7 (**+0.2**)	71.6 ± 1.0 (**+0.4**)	95.8 ± 0.4 (**-0.2**)	67.7 ± 0.5 (**+1.0**)	0.456 ± 0.005 (**+0.9**)
	GIN	78.7 ± 1.4 (**+1.3**)	67.5 ± 1.4 (**+2.4**)	96.6 ± 0.1 (**-0.1**)	55.5 ± 0.6 (**+2.8**)	0.357 ± 0.014 (**+7.5**)
QAT (W4A4)	GCN	77.2 ± 2.5	64.1 ± 4.1	70.6 ± 2.4	38.1 ± 1.6	0.692 ± 0.013
	GAT	55.6 ± 5.4	65.3 ± 1.9	76.3 ± 1.2	41.0 ± 1.1	0.655 ± 0.032
	GIN	42.5 ± 4.5	18.6 ± 2.9	91.0 ± 0.6	45.6 ± 3.6	0.572 ± 0.02
nQAT (W4A4)	GCN	78.1 ± 1.5	65.8 ± 2.6	70.9 ± 1.5	40.1 ± 0.7	0.669 ± 0.128
	GAT	54.9 ± 5.6	65.5 ± 1.7	78.4 ± 1.5	41.0 ± 0.6	0.637 ± 0.012
	GIN	45.0 ± 5.0	34.6 ± 3.8	91.3 ± 0.5	48.7 ± 1.7	0.561 ± 0.068
DQ (W4A4)	GCN	78.3 ± 1.7 (**+0.2**)	66.9 ± 2.4 (**+1.1**)	84.4 ± 1.3 (**+13.5**)	51.1 ± 0.7 (**+11.0**)	0.536 ± 0.011 (**+26.2**)
	GAT	64.4 ± 9.3 (**+9.5**)	67.6 ± 1.5 (**+2.1**)	93.1 ± 0.3 (**+14.7**)	56.5 ± 0.6 (**+15.5**)	0.520 ± 0.021 (**+20.6**)
	GIN	69.9 ± 3.4 (**+24.9**)	60.8 ± 2.1 (**+26.2**)	95.5 ± 0.4 (**+4.2**)	50.7 ± 1.6 (**+2.0**)	0.431 ± 0.012 (**+23.2**)

Table 2: This table is divided into three sets of rows with FP32 baselines at the top. We provide two baselines for INT8 and INT4: standard QAT and stochastic QAT (nQAT). Both are informed by the analysis in 4.1, with nQAT achieving better performance in some cases. Models trained with Degree-Quant (DQ) are always comparable to baselines, and usually substantially better, especially for INT4. **DQ is a stable method which requires little tuning to obtain excellent results across a variety of architectures and datasets.**

Quantization	Model	REDDIT-BIN (Acc. %) ↑
Ref. (FP32)	GIN	92.2 ± 2.3
Ours (FP32)	GIN	92.0 ± 1.5
QAT-W8A8	GIN	76.1 ± 7.5
nQAT-W8A8	GIN	77.5 ± 3.4
DQ-W8A8	GIN	91.8 ± 2.3 (**+14.3**)
QAT-W4A4	GIN	54.4 ± 6.6
nQAT-W4A4	GIN	58.0 ± 6.3
DQ-W4A4	GIN	81.3 ± 4.4 (**+23.0**)

Table 3: Results for DQ-INT8 GIN models perform nearly as well as at FP32. For INT4, DQ offers a significant increase in accuracy.

Device	Arch.	Zinc (Batch=10K) FP32	W8A8	Speedup	Reddit FP32	W8A8	Speedup
CPU	GCN	181ms	42ms	4.3×	13.1s	3.1s	4.2×
	GAT	190ms	50ms	3.8×	13.1s	2.8s	4.7×
	GIN	182ms	43ms	4.2×	13.1s	3.1s	4.2×
GPU	GCN	39ms	31ms	1.3×	191ms	176ms	1.1×
	GAT	17ms	15ms	1.1×	OOM	OOM	-
	GIN	39ms	31ms	1.3×	191ms	176ms	1.1×

Table 4: INT8 latency results run on a 22 core 2.1GHz Intel Xeon Gold 6152 and, on a GTX 1080Ti GPU. Quantization provides large speedups on a variety of graphs for CPU and non-negligible speedups with unoptimized INT8 GPU kernels.

DQ results in even larger gains compared to W4A4 baselines. We see DQ being more effective for GIN layers, outperforming INT4 baselines for Cora (+24.9%), Citeseer (+26.2%) and REDDIT-BINARY (+23.0%) by large margins. Models trained with DQ at W4A4 for graph classification and graph regression also exhibit large performance gains (of over 10%) in most cases. For ZINC, all models achieve over 20% lower regression loss. Among the top performing models using DQ, ratios of p_{min} and p_{max} in $[0.0, 0.2]$ were the most common. Figure 10 in the appendix shows validation loss curves for GIN models trained using different DQ probabilities on the REDDIT-BINARY dataset.

5 Discussion

Latency and Memory Implications. In addition to offering significantly lower memory usage (4× with INT8), quantization can reduce latency—especially on CPUs. We found that with INT8

arithmetic we could accelerate inference by up to $\mathbf{4.7}\times$. We note that the latency benefit depends on the graph topology and feature dimension, therefore we ran benchmarks on a variety of graph datasets, including Reddit[2], Zinc, Cora, Citeseer, and CIFAR-10; Zinc and Reddit results are shown in table 4, with further results given in the appendix. For a GCN layer with in- and out-dimension of 128, we get speed-ups of: $4.3\times$ on Reddit, $2.5\times$ on Zinc, $1.3\times$ on Cora, $1.3\times$ on Citeseer and, $2.1\times$ on CIFAR-10. It is also worth emphasizing that quantized networks are necessary to efficiently use accelerators deployed in smartphones and smaller devices as they primarily accelerate integer arithmetic, and that CPUs remain a common choice for model serving on servers. The decrease in latency on CPUs is due to improved cache performance for the sparse operations; GPUs, however, see less benefit due to their massively-parallel nature which relies on mechanisms other than caching to hide slow random memory accesses, which are unavoidable in this application.

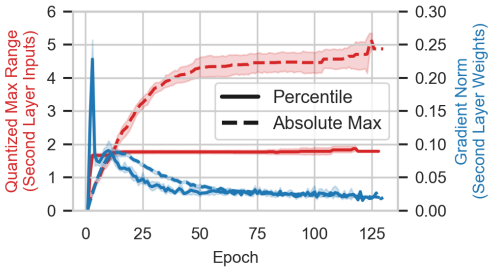


Figure 5: $q_{\max}$ with absolute min/max and percentile ranges, applied to INT8 GCN training on Cora. We observe that the percentile max is half that of the absolute, *doubling* resolution for the majority of values.

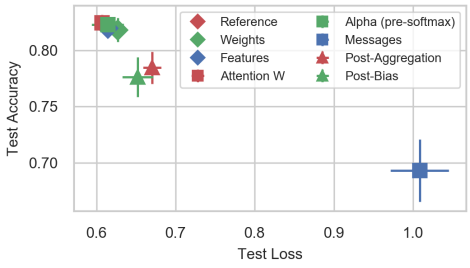


Figure 6: Analysis of how INT8 GAT performance degrades on Cora as individual elements are reduced to 4-bit precision *without DQ*. For GAT the message elements are crucial to classification performance.

Ablation Study: Benefits of Percentile Ranges. Figure 5 shows the value of percentiles during training. We see that when using absolute min/max the upper range grows to over double the range required for 99.9% of values, effectively halving the resolution of the quantized values. DQ is more stable, and we obtained strong results with an order of magnitude less tuning relative to the baselines.

Ablation Study: Source of Degradation at INT4. Figure 6 assesses how INT8 GAT (without DQ) degrades as single elements are converted to INT4, in order to understand the precipitous drop in accuracy in the INT4 baselines; further plots for GCN and GIN are included in the appendix. We observe that most elements cause only modest performance losses relative to a full INT8 model. DQ is most important to apply to elements which are constrained by *numerical precision*, such as the aggregation and message elements in GAT. Weight elements, however, are consistently unaffected.

Ablation Study: Effect of Stochastic Element in Degree-Quant. We observe that the stochastic masking in DQ alone often achieves most of the performance gain over the QAT baseline; results are given in table 9 in the appendix. The benefit of the percentile-based quantization ranges is *stability*, although it can yield some performance gains. The full DQ method provides consistently good results on all architectures and datasets, without requiring an extensive search used in section 4.1.

6 Conclusion

This work has presented Degree-Quant, an architecture-agnostic and stable method for training quantized GNN models that can be accelerated using off-the-shelf hardware. With 4-bit weights and activations we achieve $8\times$ compression while surpassing strong baselines by margins regularly exceeding 20%. At 8-bits, models trained with DQ perform on par or better than the baselines while achieving up to $4.7\times$ lower latency than FP32 models. Our work offers a comprehensive foundation for future work in this area and is a first step towards enabling GNNs to be deployed more widely, including to resource constrained devices such as smartphones.

[2]The largest graph commonly benchmarked on in the GNN literature

ACKNOWLEDGMENTS AND DISCLOSURE OF FUNDING

This work was supported by Samsung AI, Arm and, by the UK's Engineering and Physical Sciences Research Council (EPSRC) with grants EP/M50659X/1 and EP/S001530/1.

REFERENCES

R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, 2012.

Milad Alizadeh, Javier Fernández-Marqués, Nicholas D. Lane, and Yarin Gal. A empirical study of binary neural networks' optimisation. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJfUCoR5KX.

Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=BJOFETxR-.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning?, 2020.

Zachariah Carmichael, Hamed F. Langroudi, Char Khazanov, Jeffrey Lillie, John L. Gustafson, and Dhireesha Kudithipudi. Deep positron: A deep neural network using the posit number system, 2018.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets, 2020.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations, 2015.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks, 2020.

Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression, 2020.

Boyuan Feng, Yuke Wang, Xu Li, Shu Yang, Xueqiao Peng, and Yufei Ding. Sgquant: Squeezing the last bit on graph neural networks with specialized quantization, 2020.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017. URL http://arxiv.org/abs/1704.01212.

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

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017.

Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. Improving the accuracy, scalability, and performance of graph neural networks with roc. In *Proceedings of Machine Learning and Systems 2020*, pp. 187–198. 2020.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation, 2018.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, Jiyan Yang, Jongsoo Park, Alexander Heinecke, Evangelos Georganas, Sudarshan Srinivasan, Abhisek Kundu, Misha Smelyanskiy, Bharat Kaul, and Pradeep Dubey. A study of bfloat16 for deep learning training, 2019.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper, 2018.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In *Proceedings of Machine Learning and Systems 2020*, pp. 230–246. 2020.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning, 2017.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2017.

Anurag Mukkara, Nathan Beckmann, Maleen Abeydeera, Xiaosong Ma, and Daniel Sanchez. Exploiting locality in graph analytics through hardware-accelerated traversal scheduling. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-51, pp. 1–14. IEEE Press, 2018. ISBN 9781538662403. doi: 10.1109/MICRO.2018.00010. URL https://doi.org/10.1109/MICRO.2018.00010.

Gabriele Prato, Ella Charlaix, and Mehdi Rezagholizadeh. Fully quantized transformer for machine translation, 2019.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. *arXiv preprint arXiv:1911.11763*, 2019.

Tao Sheng, Chen Feng, Shaojie Zhuo, Xiaopeng Zhang, Liang Shen, and Mickey Aleksic. A quantization-friendly separable convolution for mobilenets. *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, Mar 2018. doi: 10.1109/emc2.2018.00011. URL http://dx.doi.org/10.1109/emc2.2018.00011.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

Rianne van den Berg, Thomas N. Kipf, and Max Welling. Graph convolutional matrix completion, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 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, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, Xiangjian He, Yiguang Lin, and Xuemin Lin. Binarized graph neural network, 2020.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision, 2018.

Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation, 2020.

Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: A benchmark for molecular machine learning, 2017.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert, 2019.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method, 2020.

A APPENDIX

A.1 EXPERIMENTAL SETUP

As baselines we use the architectures and results reported by Fey & Lenssen (2019) for citation networks, Dwivedi et al. (2020) for MNIST, CIFAR-10 and ZINC and, Xu et al. (2019) for REDDIT-BINARY. We re-implemented the architectures and datasets used in these publications and replicated the results reported at FP32. Models using GIN layers learn parameter ϵ. These models are often referred to as GIN-ϵ. The high-level description of these architectures is shown in table 5. The number of parameters for each architecture-dataset in this work are shown in table 6.

Our infrastructure was implemented using PyTorch Geometric (PyG) (Fey & Lenssen, 2019). We generate candidate hyperparameters using random search, and prune trials using the asynchronous hyperband algorithm (Li et al., 2020). Hyperparameters searched over were learning rate, weight decay, and dropout (Srivastava et al., 2014) and drop-edge (Rong et al., 2020) probabilities. The search ranges were initialized centered at the values used in the reference implementations of the baselines. Degree-Quant requires searching for two additional hyperparameters, $p_{\min}$ and $p_{\max}$, these were tuned in a grid-search fashion. We report our results using the hyperparameters which achieved the best validation loss over 100 runs on the Cora and Citeseer datasets, 10 runs for MNIST, CIFAR-10 and ZINC, and 10-fold cross-validation for REDDIT-BINARY.

Our experiments ran on several machines in our SLURM cluster using Intel CPUs and NVIDIA GPUs. Each machine was running Ubuntu 18.04. The GPU models in our cluster were: V100, RTX 2080Ti and GTX 1080Ti.

Model	# Layers					# Hidden Units					Residual					Output MLP				
Arch.	Cit	M	C	Z	R	Cit	M	C	Z	R	Cit	M	C	Z	R	Cit	M	C	Z	R
GCN	2	4	4	4	-	16	146	146	145	-	×	✓	✓	✓	-	×	✓	✓	✓	-
GAT	2	4	4	4	-	8	19	19	18	-	×	✓	✓	✓	-	×	✓	✓	✓	-
GIN	2	4	4	4	5	16	110	110	110	64	×	✓	✓	✓	×	×	✓	✓	✓	✓

Table 5: High level description of the architectures evaluated for citation networks (Cit), MNIST (M), CIFAR-10 (C), ZINC (Z) and REDDIT-BINARY (R). We relied on Adam optimizer for all experiments. For all batched experiments, we used 128 batch-sizes. All GAT models used 8 attention heads. All GIN architectures used 2-layer MLPs, except those for citation networks which used a single linear layer.

Model	Node Classification		Graph Classification			Graph Regression
Arch.	Cora	Citeseer	MNIST	CIFAR-10	REDDIT-BIN	ZINC
GCN	23063	59366	103889	104181	-	105454
GAT	92373	237586	113706	114010	-	105044
GIN	23216	59536	104554	104774	42503	102088

Table 6: Number of parameters for each of the evaluated architectures

For QAT experiments, all elements of each network are quantized: inputs to each layer, the weights, the messages sent between nodes, the inputs to aggregation stage and its outputs and, the outputs of the update stage (which are the outputs of the GNN layer before activation). In this way, all intermediate tensors in GNNs are quantized with the exception of the attention mechanism in GAT; we do not quantize after the softmax calculation, due to the numerical precision required at this stage. With the exception of Cora and Citeseer, the models evaluated in this work make use of Batch Normalization (Ioffe & Szegedy, 2015). For deployments of quantized models, Batch Normalization layers are often *folded* with the weights (Krishnamoorthi, 2018). This is to ensure the input to the next layer is within the expected $[q_{\min}, q_{\max}]$ ranges. In this work, for both QAT baselines and QAT+DQ, we left BN layers unfolded but ensure the inputs and outputs were quantized to the appropriate number of bits (i.e. INT8 or INT4) before getting multiplied with the layer weights. We leave as future work proposing a BN folding mechanism applicable for GNNs and studying its impact for deployments of quantized GNNs.

The GIN models evaluated on REDDIT-BINARY used QAT for all layers with the exception of the input layer of the MLP in the first GIN layer. This compromise was needed to overcome the severe degradation introduced by quantization when operating on nodes with a single scalar as feature.

A.2 DATASETS

We show in Table 7 the statistics for each dataset either used or referred to in this work. For Cora and Citeseer datasets, nodes correspond to documents and edges to citations between these. Node features are a bag-of-words representation of the document. The task is to classify each node in the graph (i.e. each document) correctly. The MNIST and CIFAR-10 datasets (commonly used for image classification) are transformed using SLIC (Achanta et al., 2012) into graphs where each node represents a cluster of perceptually similar pixels or superpixels. The task is to classify each image using their superpixels graph representation. The ZINC dataset contains graphs representing molecules, were each node is an atom. The task is to regress a molecular property (constrained solubility (Jin et al., 2018)) given the graph representation of the molecule. Nodes in graphs of the REDDIT-BINARY dataset represent users of a Reddit thread with edges drawn between a pair of nodes if these interacted. This dataset contains graphs of two types of communities: question-answer threads and discussion threads. The task is to determine if a given graph is from a question-answer thread or a discussion thread.

We use standard splits for MNIST, CIFAR-10 and ZINC. For citation datasets (Cora and Citeseer), we use the splits used by Kipf & Welling (2017). For REDDIT-BINARY we use 10-fold cross validation.

Dataset	Graphs	Nodes	Edges	Features	Labels
Cora	1	2,708	5,278	1,433	7
Citeseer	1	3,327	4,552	3,703	6
Pubmed	1	19,717	108,365	500	3
MNIST	70K	40-75	564.53 (avg)	3	10
CIFAR10	60K	85-150	941.07 (avg)	5	10
ZINC	12K	9-37	49.83 (avg)	28	1
REDDIT-BINARY	2K	429.63 (avg)	497.75 (avg)	1	2
Reddit	1	232,965	114,848,857	602	41
Amazon	1	9,430,088	231,594,310	300	24

Table 7: Statistics for each dataset used in the paper. Some datasets are only referred to in fig. 1

A.3 QUANTIZATION IMPLEMENTATIONS

In section 4.1 we analyse different readily available quantization implementations and how they impact in QAT results. First, vanilla STE, which is the reference STE (Bengio et al., 2013) that lets the gradients pass unchanged; and gradient clipping (GC), which clips the gradients based on the maximum representable value for a given quantization level. Or in other words, GC limits gradients if the tensor's magnitudes are outside the $[q_{\min}, q_{\max}]$ range.

$$x_{\min} = \begin{cases} \min(X) & \text{if step} = 0 \\ \min(x_{\min}, X) & \text{otherwise} \end{cases} \tag{1}$$

$$x_{\min} = \begin{cases} \min(X) & \text{if step } = 0 \\ (1-c)x_{\min} + c\min(X) & \text{otherwise} \end{cases} \tag{2}$$

The quantization modules keep track of the input tensor's min and max values, $x_{\min}$ and $x_{\max}$, which are then used to compute $q_{\min}$, $q_{\max}$, *zero-point* and *scale* parameters. For both vanilla STE and GC, we study two popular ways of keeping track of these statistics: *min/max*, which tracks the min/max tensor values observed over the course of training; and *momentum*, which computes the moving averages of those statistic during training. The update rules for $x_{\min}$ for STE *min/max* and STE *momentum* are presented in eq. (1) and eq. (2) respectively, where X is the tensor to be quantized and c is the momentum hyperparameter, which in all our experiments is set to its default 0.01. Equivalent rules apply when updating $x_{\max}$ (omitted).

For stochastic QAT we followed the implementation described in Fan et al. (2020), where at each training step a binary mask sampled from a Bernoulli distribution is used to specify which elements of the tensor (e.g. weights, activations) will be quantised and which will be left at full precision. We experimented with block sizes larger than one (i.e. a single scalar) but often resulted in a sever drop in performance. All the reported results use block size of one.

A.4 Degradation Studies

Figures 7 and 8 show the results of the ablation study conducted in section 5 for GCN and GIN. We observe that GCN is more tolerant to INT4 quantization than other architectures. GIN, however, requires accurate representations after the update stage, and heavily suffers from further quantization like GAT. The idea of performing different stages of inference at different precisions has been proposed, although it is uncommon (Wang et al., 2018).

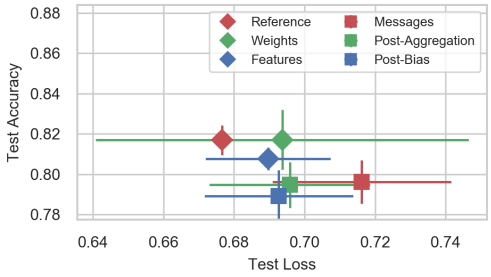


Figure 7: Degradation of INT8 GCN on Cora as individual elements are converted to INT4 *without Degree-Quant*.

Figure 8: Degradation of INT8 GIN on Cora as individual elements are converted to INT4 *without Degree-Quant*.

Quantization Scheme	Model Arch.	Node Classification Cora ↑	Node Classification Citeseer ↑	Graph Regression ZINC ↓
QAT-INT8 + DQ	GCN	81.1 ± 0.6	71.0 ± 0.7	0.468 ± 0.014
	GAT	82.1 ± 0.1	71.4 ± 0.8	0.462 ± 0.005
	GIN	78.9 ± 1.2	67.1 ± 1.7	0.347 ± 0.028

Table 9: Results obtained with only the stochastic element of Degree-Quant enabled. Percentile-based quantization ranges are disabled in these experiments.

Device	Arch.	CIFAR-10 FP32	CIFAR-10 W8A8	CIFAR-10 Speedup	Cora FP32	Cora W8A8	Cora Speedup	Citeseer FP32	Citeseer W8A8	Citeseer Speedup
CPU	GCN	182ms	88ms	2.1×	0.94ms	0.74ms	1.3×	0.97ms	0.76ms	1.3×
	GAT	500ms	496ms	1.0×	0.86ms	0.78ms	1.1×	0.99ms	0.88ms	1.1×
	GIN	144ms	44ms	3.3×	0.85ms	0.68ms	1.3×	0.95ms	0.55ms	1.7×
GPU	GCN	2.1ms	1.6ms	1.3×	0.08ms	0.09ms	0.9×	0.09ms	0.09ms	1.0×
	GAT	30.0ms	27.1ms	1.1×	0.57ms	0.64ms	0.9×	0.56ms	0.64ms	0.9×
	GIN	20.9ms	16.2ms	1.2×	0.09ms	0.07ms	1.3×	0.09ms	0.07ms	1.3×

Table 10: INT8 latency results run on a 22 core 2.1GHz Intel Xeon Gold 6152 and, on a GTX 1080Ti GPU. All layers have 128 in/out features. For CIFAR-10 we used batch size of 1K graphs.

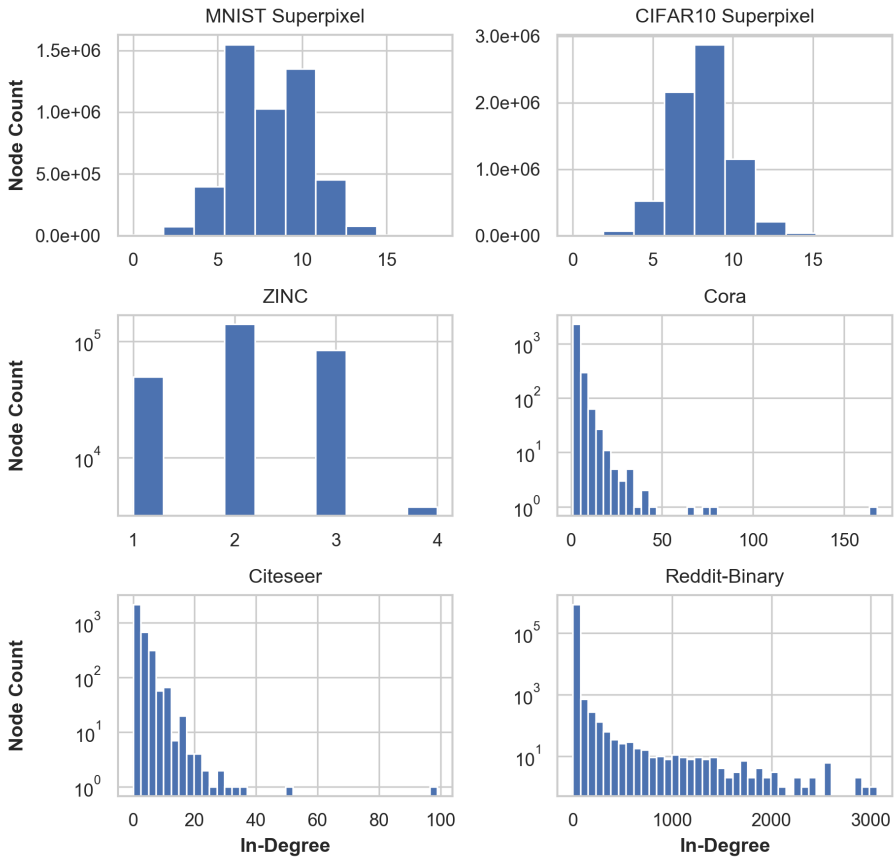


Figure 9: In-degree distribution for each of the six datasets assessed. Note that a log y-axis is used for all datasets except for MNIST and CIFAR-10.

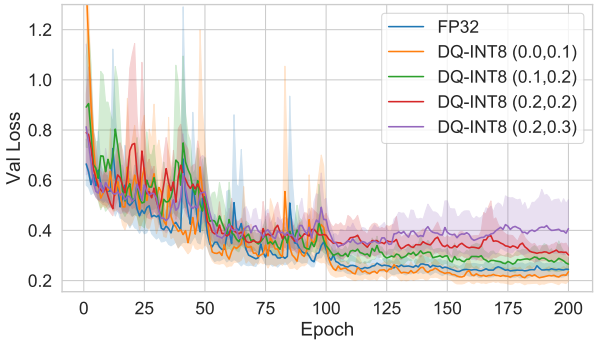


Figure 10: Validation loss curves for GIN models evaluated on REDDIT-BINARY. Results averaged across 10-fold cross-validation. We show four DQ-INT8 experiments each with a different values for ($p_{\min}$,$p_{\max}$) and our FP32 baseline.

Quantization	Model	REDDIT-BIN ↑
Ref. (FP32)	GIN	92.2 ± 2.3
Ours (FP32)	GIN	92.0 ± 1.5
DQ-INT8 (0.0, 0.1)	GIN	91.8 ± 2.3
DQ-INT8 (0.1, 0.2)	GIN	90.1 ± 2.5
DQ-INT8 (0.2, 0.2)	GIN	89.0 ± 3.0
DQ-INT8 (0.2, 0.3)	GIN	88.1 ± 3.0

Table 8: Final test accuracies for FP32 and DQ-INT8 models whose validation loss curves are shown in fig. 10