# Are GNNs fundamentally bottlenecked?

Andrei-Tiberiu Alexandru

## 1 Introduction

Most machine learning is done on unstructured data: points that are assumed to be independent and identically distributed according to some underlying probability distribution, but which are not otherwise explicitly connected. In some domains however, some of the useful information is encoded as structure – relationships between the data points, or between features of each data point. The main mathematical formalism for this type of situation is the graph, and the most widely used machine learning technique for learning on graphs is the Graph Neural Network (GNN)[1]. GNNs have proved to be effective in many situations, on different tasks and datasets such as biochemical and molecular graphs[2], citation networks[3], social networks[4], and traffic[5]. See [6] for a comprehensive survey.

One disadvantage of GNNs is that they are not straightforward to scale. In the node classification task, for example, stacking more and more GNN layers leads to over-smoothing[7], a phenomenon where the representations of all nodes converge to a common value. Another situation in which it is difficult to scale GNNs occurs when the graph contains nodes that become bottlenecks, making it impossible to transmit information from one part of the graph to another, a phenomenon known as over-squashing. There is good empirical evidence for both these phenomena, and although they may co-occur, it's likely that the over-squashing issue dominates in tasks where long-term interactions or dependencies need to be captured to solve the task.

One way to address over-squashing is by using a hybrid architecture that includes GNN layers and a transformer. This idea of hybrid architectures traces back to research in computer vision, where the approach has been fruitful, and has taught the community more about the strengths and weaknesses of transformers. In this paper, we investigate the effectiveness of such a model, GraphTrans, introduced by [8]. GraphTrans outperforms the GNNs on which it is built, and it seems like a promising approach for circumventing some of the limitations of message passing. I'm interested in whether the transformer architecture is key to the performance of GraphTrans, or if it is possible that the gains are mostly due to increased parameter count and computation performed. To do so, I compare GraphTrans against typical GNN architectures with equal amounts of parameters on the same data.

The rest of this paper is laid out as follows: section 2 covers related work and a set of preliminaries, section 3 introduces the GraphTrans architecture, sections 4 and 5 describe our experiments and their results, and the final section covers a discussion of the findings as well as opportunities for future work.

## 2 Preliminaries

### 2.1 Graph neural networks

Graph neural networks (GNNs)[1] are a mechanism for message passing in a graph. The number of iterations of message passing largely coincides with the number of GNN layers used, because each layer performs one local computation which updates a node's representation with an aggregation of its neighbour nodes' representations. The different GNN architectures differ in how the aggregation and update functions are implemented, and achieve different expressivity and efficiency trade-offs [9].

Let us briefly introduce the GNN formalism. The data is represented using a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, containing nodes $\mathcal{V}$ and edges $\mathcal{E}$, where an edge from a node $u$ to a node $v$ is denoted as $(u, v)$. Each node has an internal representation, which is initially the data itself: $\mathbf{h}_v^0 = \mathbf{x}_v$. Each GNN layer $k$ operates on the representation $\mathbf{h}_v^k$, and updates it as follows:

$$\mathbf{h}_v^k = f_k\Big(\mathbf{h}_v^{k-1}, \{\mathbf{h}_u^{k-1} | u \in \mathcal{N}_v\}; \theta_k\Big)$$

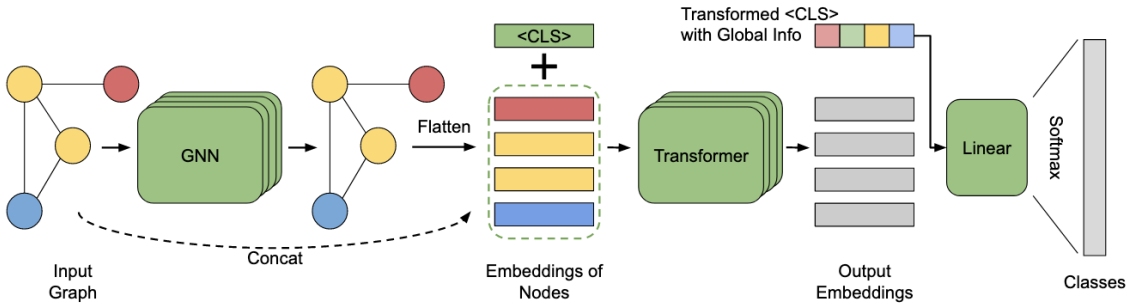


Figure 1: GraphTrans architecture, from [8].

where $\mathcal{N}_v = \{u \in \mathcal{V} | (u, v) \in \mathcal{E}\}$ is the neighbourhood of the node $v$. Similarly to a convolution, the message passing operation is local – it operates only on one node and its immediate neighbours each time it is applied. For a GNN to incorporate global information when updating a single node's latent vector, multiple layers are stacked. As more layers are stacked, each node can "see" further when it updates its internal state, and incorporates more information from across the graph. This distance is called a node's receptive field, which is defined recursively as:

$$\mathcal{N}_v^K := \mathcal{N}_v^{K-1} \cup \{w | (w, u) \in \mathcal{E} \wedge u \in \mathcal{N}_v^{K-1}\}$$

In [10], the related idea of a problem radius is introduced. A problem radius is the length of interaction between two nodes that a network must incorporate in order to solve the prediction task. For example, if there are two nodes in an input graph whose shortest path is 3, and whose interaction is relevant, then the problem radius is at least 3. (Since there could be even longer interactions.)

Using the problem radius formalism, we can consider whether some tasks have smaller or larger problem radii. A node classification task is mostly considered to have a small radius, since nodes that have the same class tend to be clustered together. For other tasks, like graph regression, a correct prediction may require factoring in long-distance interactions – so the problem radius is large.

## 2.2 Over-smoothing

Besides known limits to expressivity of different GNN architectures, there are two main phenomena that limit the performance of GNNs as more layers are added. One dominates for smaller problem radii and is called over-smoothing, and the other – over-squashing – seems to be most harmful when long-distance interactions are important.

Over-smoothing is a relatively straightforward phenomenon: as more GNN layers are stacked, the representation of each node becomes more and more like the representations of its neighbours [7]. For a task like node classification, this smoothing is useful, because it helps classify nodes with similar representations as the same class. The problem is that increasing the number of layers leads to nodes across different classes sharing the same representation, and can even lead to all nodes in a graph to become indistinguishable. Over-smoothing essentially sets a practical upper-bound for how many GNN layers can be used, and in doing so limits the performance of GNNs.

There are multiple solutions to this issue, most notably regularisation of the training objective. In [11], a metric based on the mean average distance between each pair of nodes is proposed as a target. More recently, [12] proposes adding noise to the nodes and regularising to encourage the network to map to the clean node representations. There are alternative approaches, some of which involve changing the topology of the graph based on an initial prediction. For example, [11] propose training a GNN on the original graph, then removing inter-class and adding intra-class edges according to the network predictions, then retraining the GNN. A similar approach is proposed by [7] where a GCN is trained on the gold standard labels, then retrained on the most likely labels from the initial GCN's predictions.

## 2.3 Over-squashing

For larger problem radii, over-squashing seems to cause most of the decrease in performance as more GNN layers are stacked. With extra layers, the receptive field of each node increases exponentially: first including the immediate neighbours, then neighbours' neighbours, and so on. When exponentially growing information – from very many nodes – is incorporated into a single node's fixed-size representation, a bottleneck occurs, and much of this information is lost. This disproportionately affects nodes that are further away, because they may pass through many bottlenecks to reach any given node.

Mitigation strategies for over-squashing focus on the topology of the graph itself: if a bottleneck can be removed through rewiring of the graph, the problem disappears, and we can benefit from very deep GNNs. One such approach is proposed in [10], where the final layer of a GNN is replaced with a fully adjacent layer – where every pair of nodes is connected –, thus removing the bottleneck. The downside of this approach is its complexity: $\mathcal{O}(n^2)$. In a similar vein, [13] use balanced Forman curvature, under which edges with negative values are bottlenecks and lead to over-squashing. They then target these edges specifically, and rewire them using a process called stochastic discrete Ricci flow.

Another interesting detail is that over-squashing seems to be more of an issue for GNN variants that absorb incoming edges equally, like the GCN [9] or GIN [14], but less so for graph attention networks (GATs) [15], which use an attention mechanism. This hints at the idea that using some form of attention can further alleviate over-squashing, which has prompted research into architectures that combine GNN layers with transformer layers.

## 2.4 Transformers

The transformer[16] is one of the most successful neural network architectures in the past years. Its main algorithmic innovation, multi-head self-attention is able to retain context as the length of an input sequence increases – something previous architectures, like the LSTM [17] have struggled with. Attention also abstracts away the idea of an n-gram – and the need to choose a particular context size – by being able to attend to all tokens in the input.

At least part of the transformer architecture's success is its ability to scale with more data and more parameters, leading to nothing short of a paradigm shift in the NLP community: it is now standard practice to pre-train a model on very large unlabelled datasets, then fine-tune on a smaller downstream task. In some cases, good performance can be achieved without fine-tuning, in the so-called few-shot learning scenario [18]. Generative pre-trained transformers have been behind some of the most impressive results in NLP in recent years, and there is still room for improvement [19, 20]. Transformers' success is also not limited to NLP; they have recently been applied in computer vision [21], where the hope is that they can more efficiently perform tasks that are typically associated with CNNs and ResNets [22], like image classification or object detection.

One drawback of self-attention is that the computation itself has quadratic complexity. This means that computing attention over very large inputs is costly, and may not improve on the performance of network architectures whose inductive biases are well-suited to that particular input class (like a CNN or a GNN). On the other hand, recent research has been successful in finding more efficient implementations of transformers, achieving better complexity overall [23].

## 2.5 Self-attention as drop-in replacement for inductive bias

The success of transformers in NLP has prompted researchers to apply attention to different tasks. When it comes to integrating transformers into existing architectures, there are interesting parallels between developments in the computer vision domain and in graph representation learning. The canonical architecture in CV is the convolutional neural network, whose state-of-the-art performance can (at least in part) be traced back to a set of useful inductive biases like locality and translational equivariance. Convolutions are equivariant in the sense that moving the input also moves the convolved representation. When combined with pooling operations, CNNs also exhibit translational invariance; for example, an image of a cat should not depend on where the subject is in the image.

In some sense, CNNs can be seen as networks that have a strong inductive bias; they are easy to train on the sort of data where this bias is useful (images), and not so easy on data that does not "fit" the bias (other representations). A transformer is more general – since attention happens over the entire input – and only requires that it receives a sequence. Of course, the message is not quite

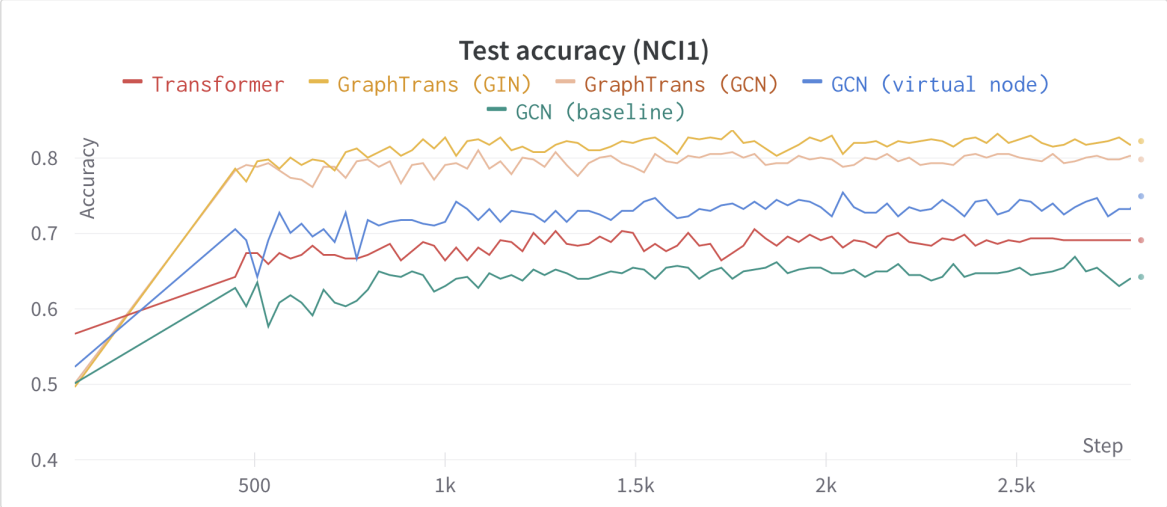


Figure 2: Baseline results for the NCI1 dataset. The models in this run have different parameter counts, which makes the comparison inadequate.

that strong inductive biases are bad, otherwise we would use simple fully-connected networks for all tasks. Rather, it seems that there is a trade-off to be made depending on availability of data and computation.

In recent years, computer vision has seen a gradual transition from strongly biased CNN-based architectures to more weakly biased transformer-based architectures. One of the most notable approaches has been simply replacing the convolutions in the later layers of a ResNet [22] with a transformer bottleneck block – a "BotNet" [24]. This strictly improved the performance of the underlying ResNet, and showed that it is possible to apply attention over relatively large inputs by first learning abstract and lower resolution feature maps (essentially using the ResNet to downsample the input).

Another interesting theoretical result along this transition was [25], which showed that in practice, multi-head self-attention can and does implement the convolution operation when transformers are applied to vision tasks. That is, transformers successfully learn the canonical way of processing images, only through attention! Finally, the transition culminated with the vision transformer (ViT) [21], a purely attention-based architecture which followed the same paradigm from NLP of pre-training on large datasets and fine-tuning on the downstream task.

Comparing the BotNet hybrid architecture with the vision transformer on image classification, what is immediately apparent is that ViT is better by quite a lot: 88.55% vs. 77.7% for the BotNet. However, what is even more illuminating is the scale of the two networks: the ViT has 632m parameters, whereas the BotNet has only 20.8m parameters. What this tells us is that all other things held equal, more data and more compute suits transformers well, and that we can get quite far without using networks with strong inductive biases. Conversely, where there is less data, or where data is highly structured, it may not be as easy to get high performance without incorporating a specific bias.

This brings us to the intersection of GNNs and transformers. Given that GNNs operate on highly structured data, should we expect that combining them or replacing them with transformers yields better performance? Or are GNNs simply very well suited to the sort of tasks we apply them to? There are two recent architectures that are almost perfect analogue to the BotNet and ViT architectures. One is GraphTrans [8], a hybrid network where a transformer is used as the "readout" function of a GNN. The other is the structure-aware transformer [26], which reformulates the attention calculation to better encode structural information into the transformer architecture. Here, we'll zoom in on the hybrid approach to see whether it is possible to use the strengths of GNNs without running into the limitations of over-smoothing and over-squashing.

# 3 GraphTrans

GraphTrans is a GNN-transformer hybrid architecture that aims to better learn long-range interactions in a graph using attention rather than additional layers of message passing. The graph is first processed by a stack of GNN layers, which produce a final encoding for each node, $\mathbf{h}_v^{L_{GNN}}$. This embedding is projected to the transformer's input dimension, normalised, and then fed into a standard transformer layer stack, without additional positional encodings (since structure would have already been learned by the GNN). Since GraphTrans is applied to the graph classification task, it must extract a single representation for the entire graph before making a prediction. It does so by collapsing the node representations into a single special-token embedding (called CLS following a convention from text classification), and then produces the predicted class via:

$$y = \text{softmax}(\mathbf{W}^{out}\mathbf{h}_{CLS}^{L_{TF}}) \tag{1}$$

Because it only assumes that there is a set of final node representations, GraphTrans is flexible – it can be built on top of most flavours of GNNs. Armed with useful learned representations and the ability to attent globally over all nodes, seemingly outperforms the GNNs it builds on, across multiple variants the authors tried: GCNs and GINs with or without virtual nodes. The GNNs on their own outperform a simple transformer network. This suggests two things: that the structure in the data is encoding a lot of information useful for the task (which is why the transformer by itself does not do well), and that there is indeed a bottleneck to the performance of GNNs (scaling them does not improve performance by much). What it does not show is that GraphTrans is conclusively better than GNNs, because it does not compare networks of the same size. Granted, the transformer itself does not account for a large proportion of the total parameter count, however a useful test would control for parameter size: GNNs with $k$ total parameters vs. GraphTrans with $k$ total parameters.

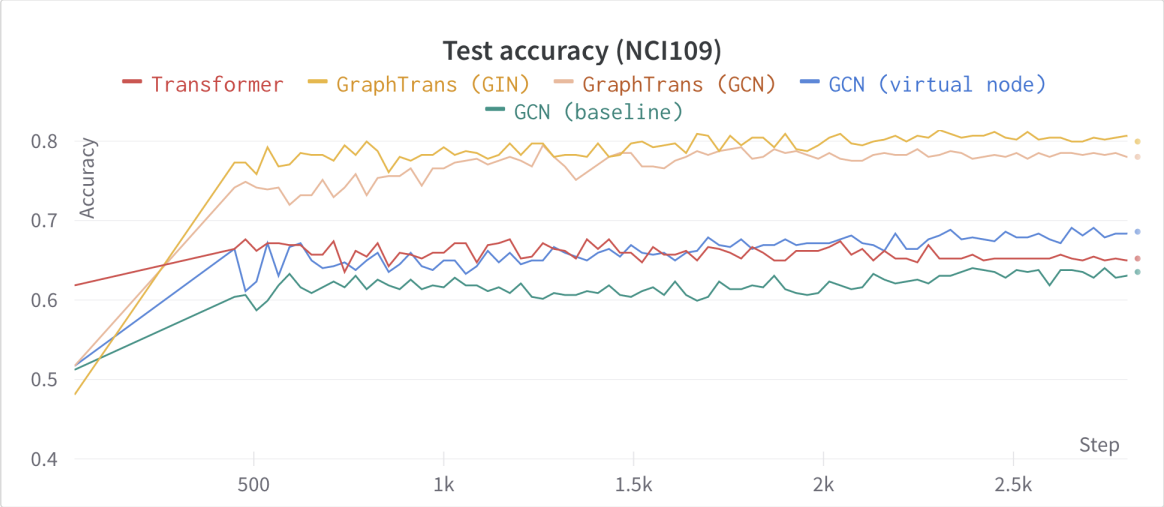


Figure 3: Baseline results for the NCI109 dataset. The models in this run have different parameter counts, which makes the comparison inadequate.

# 4 Experiments

In these experiments, I focus on the NCI[27] datasets – two relatively small datasets containing biochemical compounds – with the task of predicting whether a compound is useful for fighting lung cancer. The reason for doing so is purely practical; I don't have access to very much computational power, so I chose the smallest dataset from the original GraphTrans paper that allowed me to replicate the results.

I aimed to investigate further whether GraphTrans' increased performance when compared to the GNNs it builds on is mostly due to it being a larger model, or whether the transformer architecture

induces some qualitative change. To do so, I first replicated the NCI1 and NCI109 experiments from the original paper; the models this includes are: a plain transformer, a GCN, another GCN with a virtual node, GraphTrans built on a GCN and GraphTrans built on a GIN. I did not average multiple runs because of computational constraints.

The ablation I was most interested in is modifying the "base" GNN by increasing its number of layers, or its embedding dimension, or both. For the latter, I didn't have a good rule of thumb for how to increase the two in tandem, so the results I achieved are not necessarily the best that could be achieved at a given parameter count. I then test the scaled up models on both datasets.

I also considered doing the reverse: reducing the size of GraphTrans so that it better matched the GNNs it was built on. In practice, although I did reduce e.g. the feedforward dimension of the transformer to 32 and the number of layers in its GNN base to just 1, so that it was comparable with a 4-layer GCN, the results weren't very interesting. I think the interesting part comes with more compute and larger size; we know that transformers can scale up quite well, and we'd like to see if GNNs are so bad after all.

Briefly, the hardware used was one nVIDIA Tesla P100 GPU with 16GB VRAM, with 64 GB of DDR3 RAM and an 8-core Intel Xeon processor. Experiments are run from the official implementation of GraphTrans[8], which runs on PyTorch[28].

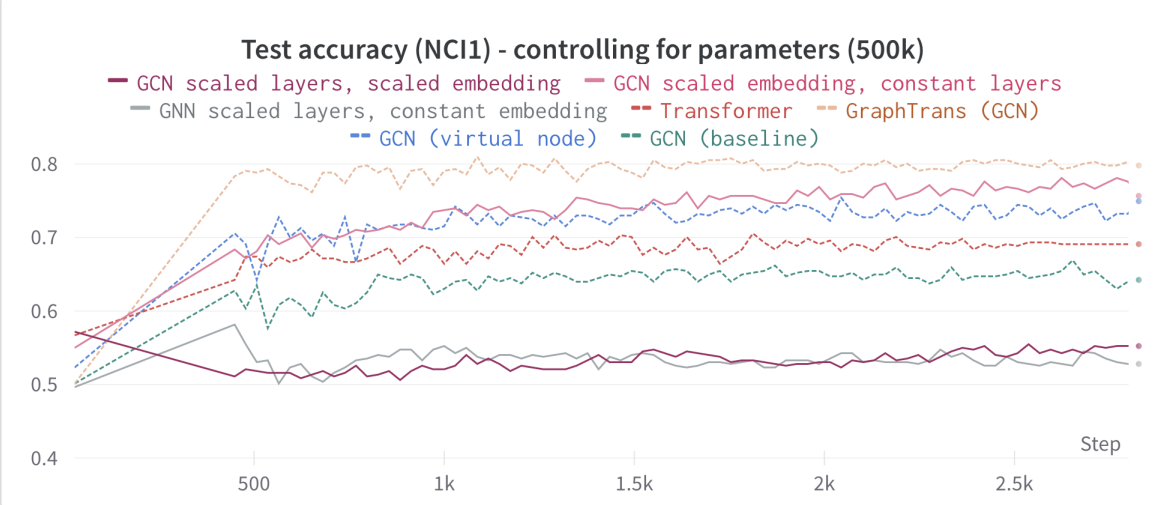


Figure 4: Results for the NCI1 dataset when controlling for model size. Here, the gap between GraphTrans and GNNs is reduced, which might mean that the gain in performance is mostly due to larger models and more compute, and not necessarily because of attention.

# 5 Preliminary results

The experiments for GraphTrans on the NCI datasets replicate. Results are in Table 1, which should be compared with Table 1 in the original paper. For the full experiment grid and a dashboard with all the results, please see [1] and [2]. It's unclear whether experiments were run with the plain GNNs, as the benchmark was against other approaches. Here I report their accuracies using configs from the official GraphTrans codebase[3], which do not control for the number of parameters. Figures 2 and 3 also show the training behaviour of all the networks in the baseline experiments.

I control for the number of parameters at two thresholds, initially. One is around 500,000 parameters, and the other around 800,000. The reason for these two values is that the baseline GraphTrans built on a GCN has the former number of parameters, and the GraphTrans built on a GIN has the

[1] https://docs.google.com/spreadsheets/d/1BUkwq7FC2f6mi--zdkuzm5AZDRObXlrBvBLGxLbSlS4
[2] https://wandb.ai/inwaves/graph-aug
[3] https://github.com/ucbrise/graphtrans/

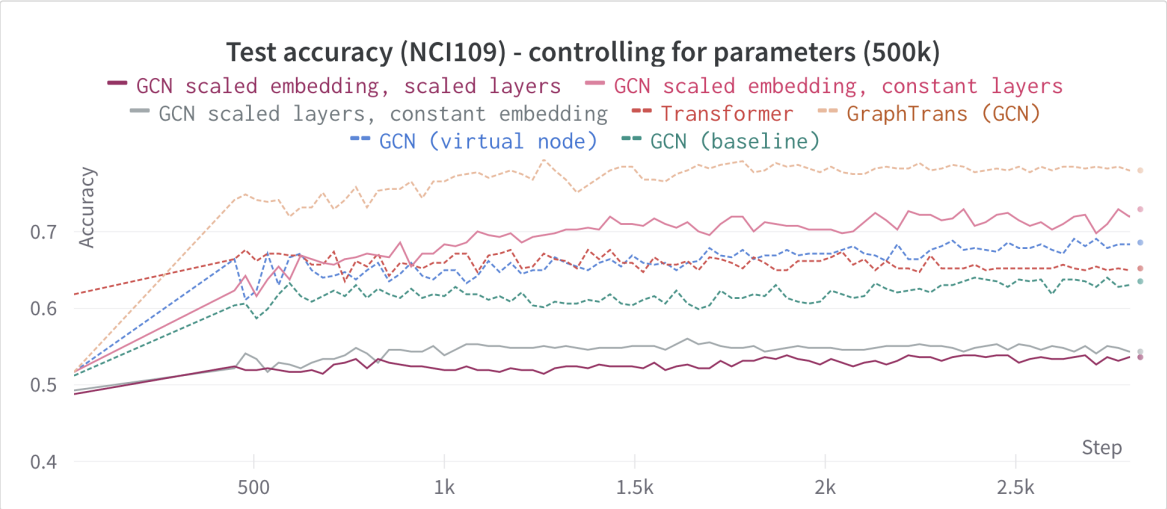


Figure 5: Results for the NCI109 dataset when controlling for model size. GraphTrans (GCN) is the most accurate model tested in this run, but it is not much better than a GCN with equal amount of parameters.

| Model | NCI1 test accuracy | NCI109 test accuracy |
|---|---|---|
| GCN | 67.1% | 66.6% |
| GCN (virtual node) | 74.2% | 67.6% |
| Transformer | 70% | 66.6% |
| GraphTrans (GCN) | 79.5% | 78.2% |
| GrpahTrans (GIN) | 82.4% | 83.4% |

Table 1: Results for the baseline experiments on GraphTrans. They are consistent with the results in Table 1 of [8], which are additionally averaged across multiple runs – these are not.

latter. (The original paper also calls them GraphTrans small and GraphTrans large, but I find this somewhat misleading because they are not the same architecture.)

When controlling for model size, a different picture emerges. Initially, the GCN was averaging about 63% accuracy against GraphTrans' 79%. When both networks contain 500k parameters, the best GCN is the architecture where the embedding dimension is increasing while holding the number of layers constant, achieving 77% accuracy. The result is robust across both datasets, although NCI109 seems a bit more challenging for all architectures. When both networks contain 800k parameters, the smaller gap is maintained: GraphTrans now yields 81% accuracy and the scaled GCN 76%. (The GCN does seem to do slightly worse, but I'm somewhat confident that averaging runs together would mostly iron this out.) Again, this applies to both datasets. Training behaviour for the first case is displayed in Figures 4 and 5, whereas the second case is in Figures 7 and 8.

It looks like GraphTrans scales better than the GCN, but that the latter can be scaled up by increasing the embedding dimension rather than the number of layers. More layers seems to improve the performance of a GCN slightly - at least on this dataset -, but much more slowly than other interventions. This points to some sort of limitation of GCNs, but it's unclear whether this is due to over-smoothing or over-squashing. It would be interesting to find out what the problem radius is for the NCI datasets, because it would shed some light on why GraphTrans works (instead of pointing to attention and repeating that it is useful).

How far can we take a meagre GCN on the NCI1 dataset? I scaled a GCN and a GraphTrans in tandem, from 800k to 1.3m and finally to 5.3m, and observed the results in Figure 6. Based on a single run, it seems that the GCN scales up fairly well, and keeps up with the transformer-GCN hybrid (though more runs should be used to make this behaviour clearer).

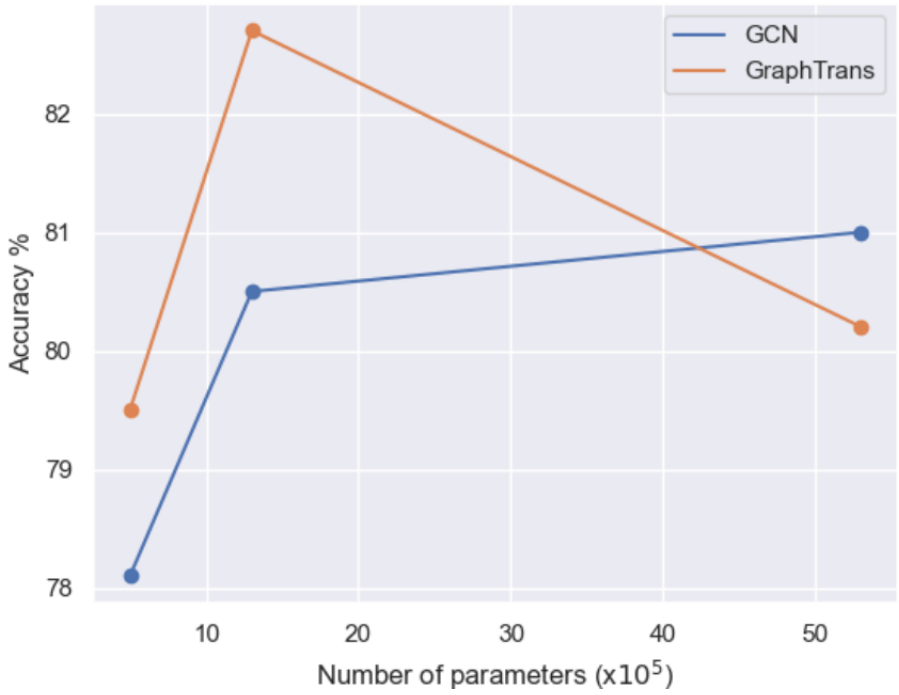


Figure 6: Accuracy scaling with model size for GCNs and GraphTrans (based on GCNs). The trend seems to be reversed as the number of parameters exceeds 5 million, with GraphTrans scaling less efficiently. This could be an artefact of the run, however.

## 6 Conclusion

GraphTrans indeed seems to be immune to some phenomenon that affects how well GNNs scale up – though it is unclear why this occurs. It is a flexible architecture that can be built on top of any GNN architecture, and adapted for different tasks. That said, the performance gap between a graph-transformer and a GNN of comparable size is not very large, which suggests that a large portion of the gains comes from simply scaling up the number of parameters.

These results are limited, and it would be premature to draw a conclusion from applying the networks to just one dataset. Further work can be done in at least two directions, both of which require more compute: (a) compare the networks on the other two datasets in the GraphTrans paper [8], and perhaps on other large datasets from the Open Graph Benchmark [29] and (b) scaling up the networks in Figure 6 even further, to see whether they indeed come apart as they become larger.

However, we can tentatively answer that GNNs are *not* fundamentally bottlenecked, and that there are ways to make them scale well to many parameters, on large datasets. There are ways in which scaling leads to oversmoothing and oversquashing, but these can be mitigated through regularisation and careful rewiring of the input graph. These strategies are worth refining, as they ultimately unlock the many benefits of depth that have characterised recent advances in machine learning.

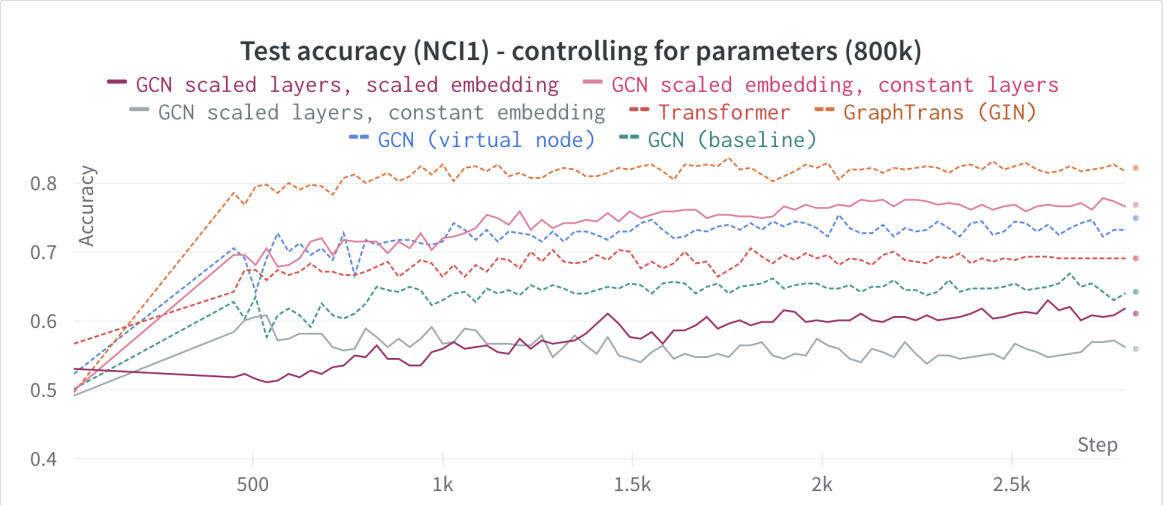


Figure 7: Results for the NCI1 dataset when controlling for model size. The GIN-based GraphTrans still leads in accuracy, but not too far off is the GCN with scaled-up embedding dimension. Other scaling methods are unsuccessful.

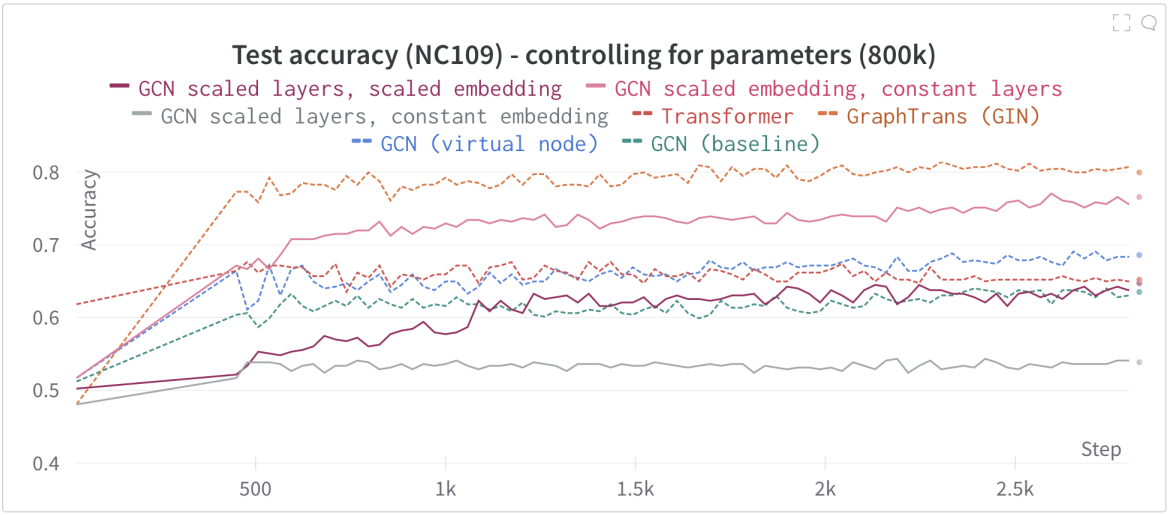


Figure 8: Results for the NCI109 dataset when controlling for model size. The behaviour is consistent with other runs on NCI109 at 500k parameters, as well as the runs on NCI1.

# References


[1] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[2] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*, pp. 1263–1272, PMLR, 2017.

[3] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The world wide web conference*, pp. 2022–2032, 2019.

[4] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax.," *ICLR (Poster)*, vol. 2, no. 3, p. 4, 2019.

[5] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *arXiv preprint arXiv:2101.11174*, 2021.

[6] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[7] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[8] Z. Wu, P. Jain, M. Wright, A. Mirhoseini, J. E. Gonzalez, and I. Stoica, "Representing long-range context for graph neural networks with global attention," *Advances in Neural Information Processing Systems*, vol. 34, pp. 13266–13279, 2021.

[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[10] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," *arXiv preprint arXiv:2006.05205*, 2020.

[11] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3438–3445, 2020.

[12] J. Godwin, M. Schaarschmidt, A. L. Gaunt, A. Sanchez-Gonzalez, Y. Rubanova, P. Veličković, J. Kirkpatrick, and P. Battaglia, "Simple gnn regularisation for 3d molecular property prediction and beyond," in *International Conference on Learning Representations*, 2021.

[13] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," *arXiv preprint arXiv:2111.14522*, 2021.

[14] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," *arXiv preprint arXiv:1810.00826*, 2018.

[15] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners,"

[19] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[20] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, *et al.*, "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, 2022.

[21] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[23] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *arXiv preprint arXiv:2009.06732*, 2020.

[24] A. Srinivas, T.-Y. Lin, N. Parmar, J. Shlens, P. Abbeel, and A. Vaswani, "Bottleneck transformers for visual recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16519–16529, 2021.

[25] J.-B. Cordonnier, A. Loukas, and M. Jaggi, "On the relationship between self-attention and convolutional layers," *arXiv preprint arXiv:1911.03584*, 2019.

[26] D. Chen, L. O'Bray, and K. Borgwardt, "Structure-aware transformer for graph representation learning," *arXiv preprint arXiv:2202.03036*, 2022.

[27] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, 2008.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[29] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *Advances in neural information processing systems*, vol. 33, pp. 22118–22133, 2020.