Exploring Techniques for Addressing Long-Range Interactions in Graph Neural Networks: A Comparative Study of Positional Encoding, Edge Creation, and Attention

Yichi Zhang, Brandon Liu

ABSTRACT

In this paper, we apply a set of transformer-based techniques to various Graph Neural Network (GNN) architectures and evaluate their effectiveness in capturing long-range interactions in the graph. Our proposed methods encompass positional encoding, edge creation, and graph attention and we examined for their performances both cooperatively and comparatively. These methods aim to address the limitations of long-range interactions between nodes in the graph structure, which can affect the quality of results obtained by GNN models. Our results suggest that while the effectiveness of each technique depends on the specific dataset and task, a joint application of transformer-based methods can significantly improve GNN models' ability to capture long-range interactions.

1 INTRODUCTION

The Graph Neural Network (GNN) is a deep learning architecture designed for learning graph-structured data, which uses nodes and edges to represent information. Compared to tabular data, the graph data are usually characterized by its lack of sequential orders, thus resulting in challenges to apply common deep learning techniques on them. The standard approach for processing GNNs is to perform message passing between directly connected nodes, and via repetitively aggregating features through neighbouring nodes and edges, the information could diffuse from further away throughout the graph.

While GNNs are generally capable of propagating information and capture the relationships between nodes, they can struggle to effectively capture long-range interactions in large graphs. Ideally, a GNN of N layers can aggregate information from nodes N steps away. However, as the number of nodes grows larger, the number of steps to traverse and the amount of information to be encoded into one vector increase exponentially. Another concern is that a considerable amount of the information can be lost during the message passing process, typically at nodes with few connected edges where the information are more compressed. This issue is known as the oversquashing issue((Alon and Yahav, 2020) and remains as one of the most common optimization issues for GNNs. Many previous works have tried to address the long-range interactions by proposing new types of GNN operations, such as positional encoding, graph attention and adding additional edges at “bottleneck” nodes, while each approach results in additional computational costs. This can make it difficult to train GNNs on very large graphs and to scale them to real-world applications.

In this paper, we explored behaviors of different constructions for capturing interactions in long-range graphs. The datasets we used include Pascal-VOC-SP, Peptides-func and Princeton Shape Benchmark(PSB), with first two come from the Long Range Graph Benchmark(LRGB) collections(Dwivedi et al., 2022) and the latter one being a standardized dataset for testing shape matches(Shilane, Philip, et al., 2004). While the ordinary practices of the GNN do not necessarily require distinguishing long-ranged graph dependencies, the benchmark datasets are believed to demand such identification to achieve a satisfying performance, which makes them ideal for the context of this paper. We’ve excerpted a table from the original dataset overview to demonstrate the scope of the work (Table 1). Among the two datasets, PascalVOC-SP deal with node-level superpixel predictions and uses the marco F1 score as the performance metric. On the other hand, the task of Peptide-func PSB datasets

are to determine the category of the graph based on its global-level structure, and the average precision or mean average precision scores are used to evaluate the model performance.

Dataset	Task	Num of Graphs	Avg Num of Nodes	Metric
PascalVOC-SP	Node Prediction	11355	479.40	macro F1
Peptides-func	Graph Classification	15535	150.94	Average Precision
PSB	Graph Classification	2326	4436.61	mean Average Precision

Table 1. Scales and Tasks for Benchmark Datasets

2 METHOD SUMMARY

2.1 Datasets and Types of Tasks

We used the cross entropy as the loss function for both node prediction and graph classification task. And for node prediction task specifically we tuned the cross entropy to also incorporate class weights to address the fact that the PascalVOC-Sp dataset contains 60~70% of 0 label for each graph which constitutes to a unbalanced label distribution. Therefore, we wish to re-weight the loss function to prevent overwhelming false negative predictions. The seemingly low f1 score also reflected the imbalanced nature of this dataset.

Apart from PascalVOC-SP and Peptides-func which are well-organized dataset dedicated to graph data testing, we also employed the Princeton Shape Benchmark dataset, a standardized dataset for 3d model shape recognition, as a typical custom dataset from other domain to

testify our methods. The primitive PSB dataset store data in the format of .off files rather than graph data. The format represents a mesh composed of vertices, edges, and faces. The file begins with a header line specifying the number of vertices, edges, and faces, and is commonly used for representing 3D models in computer graphics. By parsing the vertices and faces into nodes and shape edges to graph edges, we can easily create a transformed PSB dataset for graph data. While the original PSB uses hierarchy categorization classes, e.g. “aircraft/airplane/F117”, in this experiment we would only uses the first hierarchy as the “super class” for the categorization task, as the nature of this experiment is not to test classification algorithm, but to investigate the long-rage graph interactions.

For graph-level classification, the nature of the task reveals a loosing demand on controlling class weight, and adding node-wise weight does not produce a visible influence on the result. Therefore, we used the generic cross entropy loss for evaluating graph-level tasks. To get the graph-level representation, we applied a mean pooling layer to aggregate node features from the graph, followed a 3-layer MLP classifier for the final classification. Since the goal of this project is to evaluate the graph neural network performance comparatively, the MLPs are not fine-tuned individually for different GNN structures in addition to the standard training workflow.

2.2 GCN Models

We would test our transformer-based approach across some of the most widely used GNN architectures, which are Graph Convolutional Network (GCN), Graph Isomorphism Network (GIN), Graph Attention Network (GAT), gated Graph Convolutional Network (GatedGCN), and Spectral Attention Network (SAN). While the focus of this paper is transformer-based methods for GNNs, understanding the overview of the used GNN model pipelines is also

important for contextualizing the transformer-based approaches. A GNN model typically consists of several key components, such as graph convolutional layers, attention mechanisms, and gating mechanisms. These components can be modified and combined in different ways to create different GNN models, including GCN, GIN, GAT, gatedGCN, and SAN.

In traditional GCN models, information is passed between neighboring nodes through message propagation, which limits the model's ability to capture dependencies among distant nodes. However, by incorporating transformer-based approaches, the GCN models can aggregate information from a much larger range of nodes, thus improving their ability to capture long-range interactions. By understanding the components and architecture of these GNN models, we can better appreciate the specific challenges that arise in capturing long-range dependencies, which is the focus of transformer-based methods. Additionally, many of the transformer-based approaches build on existing GNN models or utilize similar components, so understanding the GNN pipeline is essential for understanding how transformer-based methods improve upon these existing models.

GCN

The Graph Convolutional network follows the path

$$H^{(k)} = ReLu(W^{(k)}H^{(k-1)}Q_{GCN}),$$

where $Q_{GCN} = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}$ marks the graph Laplacian and $W^{(k)}$ denotes the trainable parameters in the layer. Innately a spectral method, GCN is characterized by its inclination to local features. In GCN, each node aggregates information from its neighbours through a linear transformation followed by a nonlinear activation function, resulting in a new representation for the node. The final representation of each node is obtained by combining

its original representation with its aggregated representation from the neighbours(Kipf and Welling, 2019).

GIN

The Graph Isomorphism Network (GIN) takes the form

$$H^{(k)} = MLP^{(k)}(W^{(k)}H^{(k-1)}Q_{GCN}),$$

using a multiplayer perceptron for mapping the features (Xu et al., 2019), improved from GCN architectures. Unlike GCN, GIN can handle graph isomorphism, which means it can recognize that two different graphs are actually the same in structure. GIN uses a learnable multi-layer perceptron (MLP) to aggregate information from neighbors, and applies the same MLP to each node in the graph.

GAT

The Graph Attention Networks operates the pair-wise attention scores to weight the importance of neighbors during message passing(Brody et al., 2021). It computes attention coefficients between pairs of nodes using a learnable function and uses these coefficients to weight the hidden representations of the neighbors during aggregation. The GAT is suited for our experiment because it relies on the attention mechanism locally without the spectral method. The addition of other transformer-based approaches such as positional encoding and edge creating are expected to enhance its expressiveness. Moreover, we can contrast the transformer-incorporated performance of GAT to those of GCN and GIN to investigate how does attention mechanism compares to other transformer-based methods.

GatedGCN

GatedGCN is a variation of GCN that uses gated recurrent units to model temporal dependencies during message passing(Li, Yujia, et al, 2015). It computes the hidden

representation of each node by combining its previous hidden representation with its current aggregated representation from the neighbors. GatedGCN can effectively model the temporal dynamics of graphs and has shown strong performance on various graph-related tasks.

SAN

The Spectral Attention Network (Kreuzer et al. 2021) applies a positional encoding to the graph data through the whole Laplacian spectrum and passes the positional encoding to a all-connected Transformer layer alongside node features. SAN also uses a learnable Laplacian positional encoding scheme to encode the structural information of the graph. SAN has shown strong performance on the benchmark datasets and can effectively capture long-range dependencies in graphs.

2.3 Transformer-based Approaches

In our experiments, we examined the use of positional encoding, edge creation, and graph attention techniques, both independently and in combination. We will evaluate the performance of these approaches on the PascalVOC-SP and Peptide-func PSB datasets, using the macro F1 score and average precision score as the evaluation metrics, respectively. Our results show that a joint application of these transformer-based techniques can significantly enhance GNN performance in capturing long-range interactions on graphs.

Some of the GNN architectures that we used for testing have a built-in implementation of transformers, for instance, in SAN Laplacian positional encoding and full graph attention are used. In this case, we'd modified the existing implementation instead of adding new layers of transformers. We did not, however, modify the baseline model for each model type for the purpose of comparisons.

Positional Encoding

We use Laplacian positional encoding and random walk encoding as the two approaches for incorporating graph structure information into the node embeddings in graph neural networks.

Positional encoding injects the relative position of nodes into the node features, which can be generated using the Laplacian matrix of a graph. Given the Laplacian matrix $L = D - A$, where D is the diagonal matrix of node degrees and A is the adjacency matrix, we can calculate the eigenvectors from it. The Laplacian eigenvectors can then be normalized and concatenated to for a vector representation for each node in the graph. Positional encoding can provide additional information about the global structure of the graph through local node features, allowing most GNNs to handle irregular graph structures.

Random walk encoding, on the other hand, is based on the idea of simulating random walks on the graph and using the resulting probabilities to encode information about the local and global structure of the graph. Specifically, for each node in the graph, the random walk positional encoding generates a vector of probabilities that correspond to the likelihood of the node being reached by random walks of various lengths starting from every other node in the graph. These probability vectors are then used as additional features for each node in the GNN.

Positional Encoding can be added to existing GNN architecture with minimal modifications and does affect the number of trainable parameters which otherwise could increase the training complexity. The process of positional encoding itself, however, demands heavy computations. Thus, one of the goals in our experiment is to test whether it is efficient to add positional encoding above other shortcut approaches.

Edge Creation

The oversuqashing issue is typical for large graphs because both the frequencies of bottleneck cases and the overall distance between node increase as the total number of nodes increase (Dwivedi et al., 2022). Adding supplemental edges can potentially decreases the level of information compression at certain nodes, and thus mitigating the oversquashing problem. To determine which node to add edges, we use the probability weight $p = a / d(i,j)^2$ where a is a learnable parameter. And we use an overall dropout rate as a marginal limit to determine the maximal proportion of edges to be added.

Creating new edges address the long-range interactions at the cost of increasing message passing computations. However, as the message passing process is one of the fundamental unit integrated in the current GNN code base, it's computation is mostly optimized and scalable, which means it could less susceptible to the addition of computing complexity as compared to the previous two approaches.

Since the number of edges to add is entirely dependent on the specific structure of a graph, both the effectiveness and the drawbacks of this approach can be difficult to measure. On the other hand, positional encoding is an approach that reduces the structure sensitivity and can be feasibly combined with most architectures. Therefore, we combined two approaches to test if any synergized effects can be produced.

Graph Attention

Graph attention is a mechanism used in GNNs to allow nodes to attend to different neighbors based on their importance, rather than aggregating information from all neighbors equally

during the normal propagations. This can help to address the issue of long-range interactions in graphs, as it allows nodes to attend to distant neighbors that may have a greater impact on their representation. Throughout the experiment, we developed different approaches of constructing the graph attention. For choosing which node pairs to generate attention scores, we tried random selection(with learnable parameters) and weighted selection by distance using $p = a / d(i,j)^2$ as discussed in the "Edge Creation" section.

In graph attention, there are two main approaches for determining attention scores: global attention and partial attention. The choice between these two approaches will depend on the specific task and graph being considered, and the desired trade-off between accuracy and computation time. Given the scope of dataset sizes and graph sizes in our tests, global attention is computationally expensive. While it's believed that the substitutes for partial attention might be outperformed by full graph attention, given the scope of dataset sizes and graph sizes in our tests, full attention is computationally expensive and is sometimes not affordable. Using partial attention allows a reasonably run-time mitigation, especially when combined with other computationally heavy transformers, while preserving the advantage of attention scores in storing the global structure of a graph. Hence, our primary focus is the partial attention.

3 RESULT

While the transformers-based methods pose additional computational cost upon models, the inclusion of these approaches in GNN models can significantly reduce the potential training time required to achieve state-of-the-art performance, which otherwise must be achieved by adding more layers. This is particularly important in large-scale graphs, where the number of nodes and edges can be prohibitively large that disallows continuing adding layers. By

reducing the computational burden, transformer-based approaches allow for the development of more efficient and scalable GNN models, enabling the characterization of long-range interactions in even larger graphs. Table 2 to Table 4 summarizes our findings comparing the model which generates the best metrics to the baseline model for each model type. See appendix for a full collection of training result from all combinations of transformer-based method.

3.1 Metric Performance

Our modified transformers have enabled a significant reduction in training time for models compared to the previous results in LRGB. The Spectral Attention Network, which originally had two spectral mechanisms causing high computational burden, now requires only 1.7 hours on average to run with improved performance over the LRGB(Dwivedi et al., 2022). The training time has been reduced from around 60 hours to an average of 2-3 hours. While it's believed that the substitutes for partial attention might be outperformed by full graph attention, it allows a reasonably run-time mitigation when combing with other computationally heavy transformers, while preserving the advantage of attention scores in storing the global structure of a graph.

In general, the experimental findings suggest that the use of additional transformers can lead to improved performance on all the testing datasets. The results reveal that the inclusion of partial graph attention to the GNN leads to a more notable enhancement in model performance as compared to the performance gains from the utilization of positional encoding and edge creation techniques solely. For PSB, the GNN shows a strong improvement from results of using shape descriptor functions in Shilane's original paper, which ranges 0.213 for all-label classification to at best 0.416 for classifying one specific

category (Shilane, Philip, et al., 2004). It's expected that the addition of transformers also yields relatively the greatest improvements on PSB than other 2 datasets, as the PSB had more nodes and thus demands more handling to capture the long-range interactions.

As the benefits of the transformer-based approaches vary depending on the specific task and graph being considered, we've also witnessed some minor discrepancies from our results. On the Pascal-VOC-SP dataset, for instance, the performance increase by adopting partial attention in replacement of full graph attention is less significant. On the other hand, for PSB dataset, the edge creation techniques does not improve the performance, since the PSB is a dataset used for 3d shape recognition and by its nature it had almost twice as much nodes from the Pascal and Peptides datasets, the addition of artificial edges could possibly impair the recognizable shape of the existing 3d meshes, revealing the circumstantial nature of this method.

	Pascal-VOC-SP	
Model	**Metric(macro-F1)**	**Time**
base-SAN	0.31998	4596
edge-lap-SAN	0.333113	4646
base-GAT	0.210278	4474
partial-edge-walk-GAT	0.212819	6471.18
base-GIN	0.105413	4483
lap-GIN	0.11533	6225.81
base-GatedGCN	0.0998135	9774.1
lap-GatedGCN	0.103007	22086.9

base-GCN	0.0710746	3725
edge-lap-GatedGCN	0.105592	5435.82

Table 2. baseline vs. best model on Pascal-VOC-SP. For abbreviation: “base” stands for baseline models, “edge” stands for edge creation, “lap” stands for Laplacian positional encoding, “walk” stands for Random Walk Positional Encoding and “partial” stands for the partial graph attention.

Peptide-func

Model	**Metric(AP)**	**Time**
base-SAN	0.713322	3923
edge-walk-SAN	0.781493	6353.2
base-GAT	0.567336	3779
partial-walk-GAT	0.613628	5924.79
base-GCN	0.54628	3137
lap-GCN	0.597863	4429.61
base-GIN	0.588791	3192
walk-GIN	0.68621	4559.98
base-GatedGCN	0.547136	8486.34
lap-GatedGCN	0.599499	18008.5

Table 3. baseline vs. best model on Peptides-func

Princeton Shape Benchmark

Model	Metric(mAP)	Time
base-SAN	0.491718	604.534
lap-SAN	0.69522	303.984
base-GAT	0.26137	550.336
partial-lap-GAT	0.415871	311.685
base-GCN	0.226104	257.307
lap-GCN	0.39457	255.812
base-GIN	0.322955	482.445
lap-GIN	0.473845	263.543
base-GatedGCN	0.244856	967.595
lap-GatedGCN	0.348001	1509.78

Table 4. baseline vs. best model on PSB

3.2 Run Time Efficiencies

As previously mentioned, the transformer-based approaches we explored include attention mechanisms, positional encodings, and edge creation. We found that incorporating these methods into the GNN pipeline resulted in significant improvements in accuracy compared to the baseline models. To address the trade-offs between improvements and the computational complexity, we evaluated the performance of the transformer-based approaches based on their training time and eventual metrics.

In figures below, we presented a contrast between our transformer-based methods and the originally unchanged model which we would call them baseline. While the actual efficiency varied depending on the specific dataset and the particular model type, in general we could summarize that the set of transformers which best improves the models do not simultaneously result in a significantly unacceptable run-time increase.

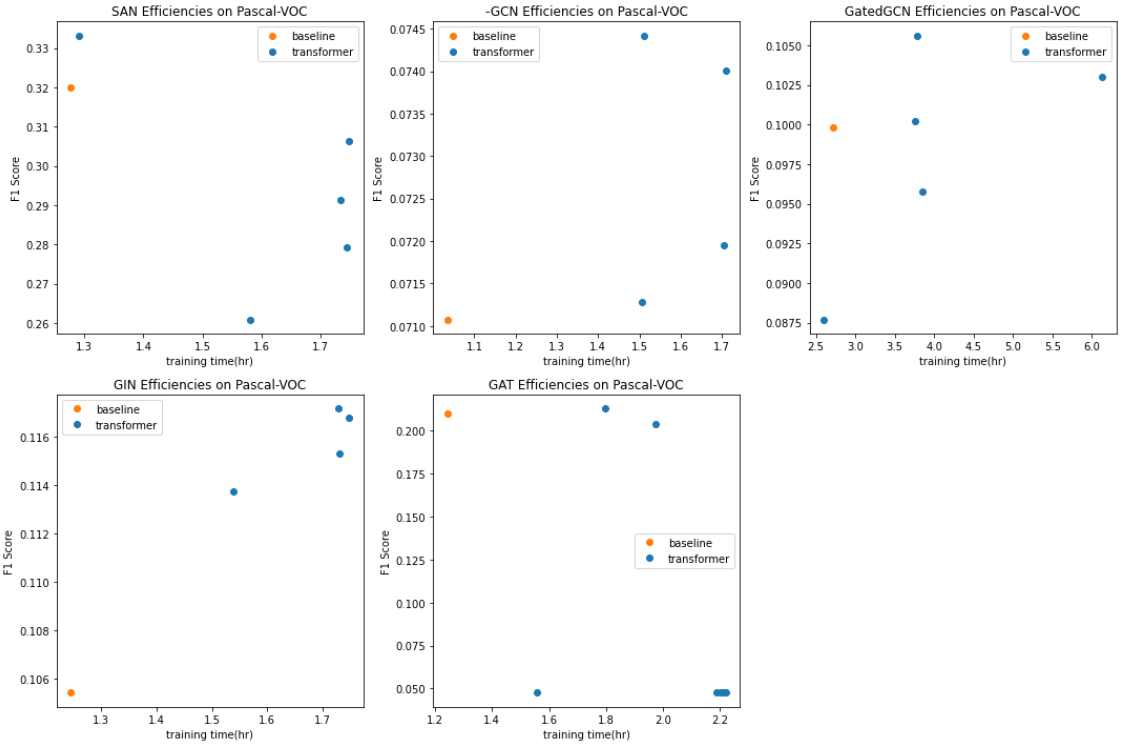

Figure 1-1. Metric performance vs. run-time efficiency on Pascal-VOC-SP

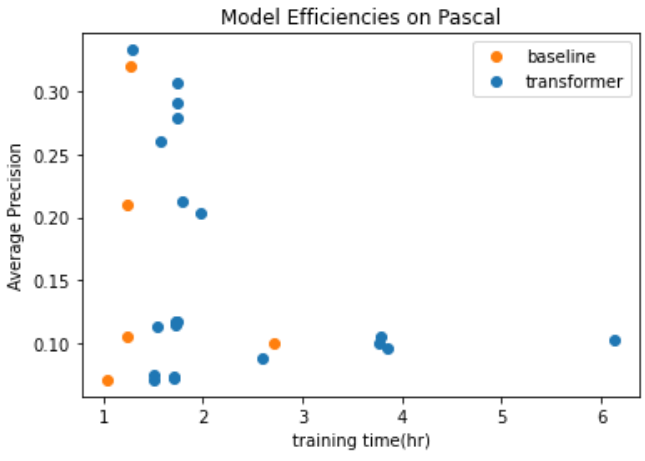

Figure 1-2. Joint model efficiencies on Pascal

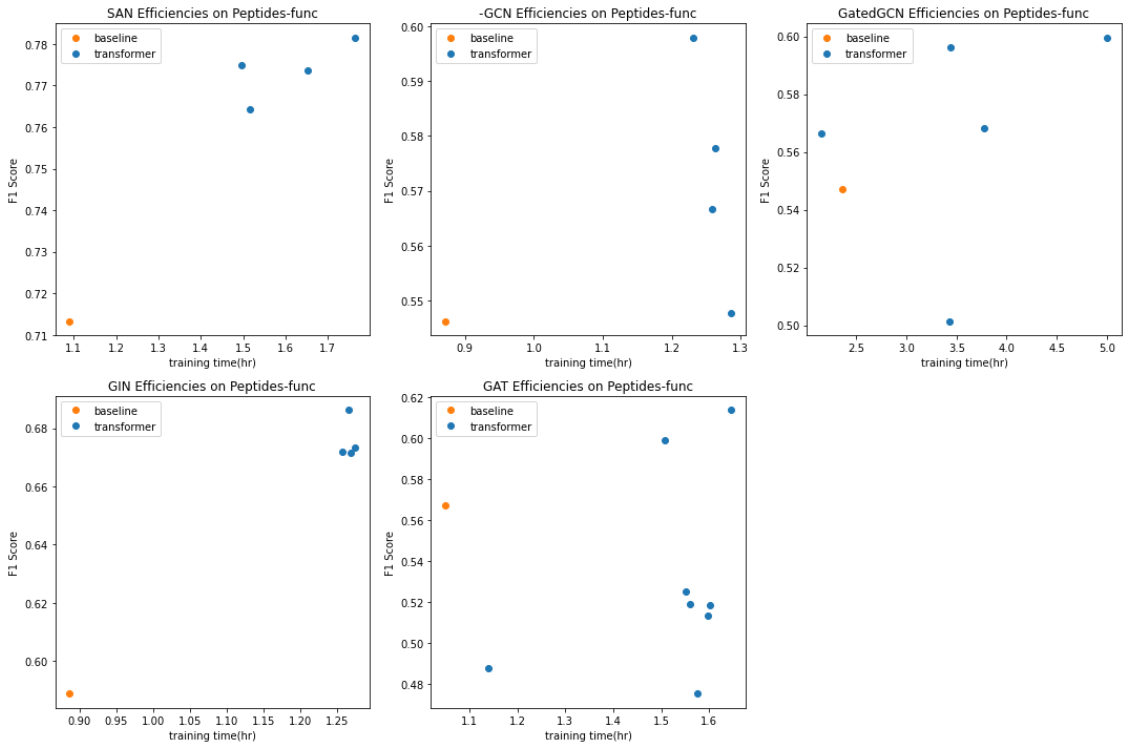

Figure 2-1. Metric performance vs. run-time efficiency on Peptides-func

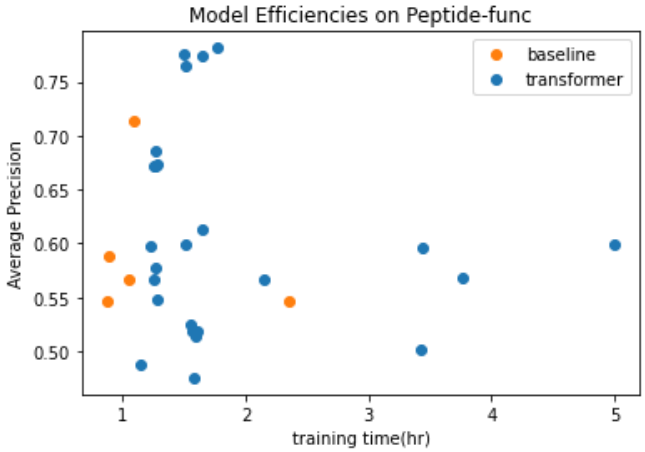

Figure 2-2. Joint model efficiencies on Peptides-func

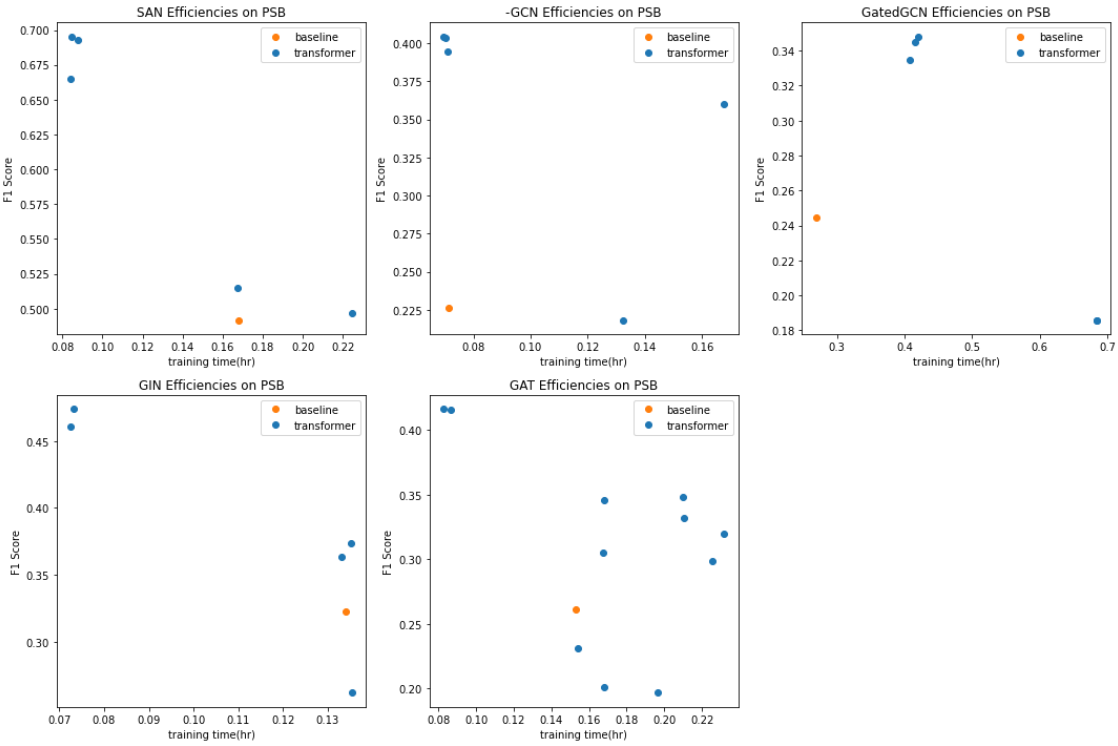

Figure 3-1. Metric performance vs. run-time efficiency on Princeton Shape Benchmark

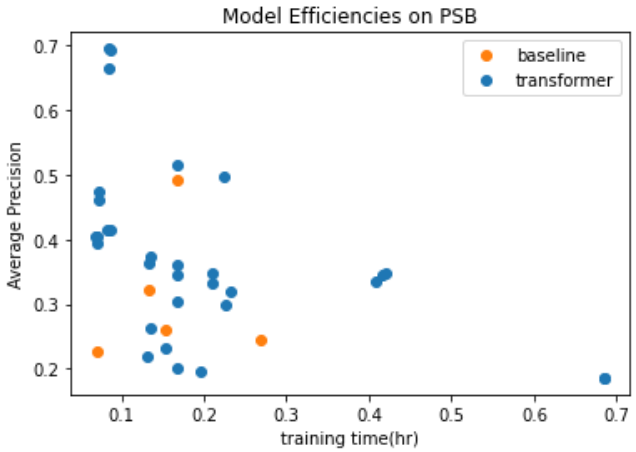

Figure 2-2. Joint model efficiencies on PSB

It's worth noting that the relative improvement of model performance on the Pascal-VOC dataset is considerably lower than on the Peptides-func dataset. This may be due to differences in the characteristics of the graphs and structures. Additionally, our modified

partial attention mechanism and positional encoding yields a negative impact on the Pascal-VOC dataset for SAN models, though the edge creation contribute to a positive effect. The reason for such odds is not clear at the current stage, and further investigations are required to conclude a plausible reason beyond dataset specificity.

4 CONCLUSION

In conclusion, we have explored the effectiveness of various transformer-based approaches in improving the performance of graph neural networks. Our findings suggest that incorporating transformer-based approaches can lead to more efficient and scalable GNN models that can effectively capture long-range interactions in larger graphs. Our result shows that the addition of partial graph attention can significantly increase the model performance more than positional encoding and edge creation does. Combining positional encoding and/or edge creation with attention can further enhance the performance by capturing both local and global structural information. Moreover, the transformer-based approaches significantly reduce the training time required to achieve state-of-the-art performance. Future research could explore on the generalization capabilities of transformer-based GNNs to new and unseen graphs.

APPENDIX

model_type	metric	model_type	metric	model_type	metric
psb-lap-SAN	0.6952196	pascal-edges-SAN	0.333113	peptides-edge-walk-SAN	0.781493
psb-edge-lap-SAN	0.6926008	pascal-base-SAN	0.31998	peptides-lap-SAN	0.774999
psb-walk-SAN	0.6653228	pascal-lap-SAN	0.306362	peptides-walk-SAN	0.773616
psb-edge-SAN	0.5145369	pascal-edge-lap-SAN	0.291276	peptides-edge-lap-SAN	0.764344
psb-edge-walk-SAN	0.4964911	pascal-walk-SAN	0.27928	peptides-base-SAN	0.713322
psb-base-SAN	0.4917182	pascal-edge-walk-SAN	0.260857	peptides-walk-GIN	0.68621
psb-lap-GIN	0.4738446	pascal-edge-GAT	0.215543	peptides-edges-SAN	0.680472
psb-edge-lap-GIN	0.461029	pascal-edge-walk-GAT	0.212819	peptides-lap-GIN	0.673308
psb-walk-GAT	0.4161945	pascal-base-GAT	0.210278	peptides-edge-lap-GIN	0.671955
psb-lap-GAT	0.4158707	pascal-walk-GAT	0.204051	peptides-edge-walk-GIN	0.671453
psb-walk-GCN	0.4042956	pascal-lap-GAT	0.157617	peptides-walk-GAT	0.613628
psb-edge-walk-GCN	0.4036116	pascal-edge-lap-GAT	0.154941	peptides-lap-GatedGCN	0.599499
psb-lap-GCN	0.3945698	pascal-edge-lap-GIN	0.117174	peptides-edge-walk-GAT	0.598884
psb-edge-walk-GIN	0.3740256	pascal-walk-GIN	0.116813	peptides-lap-GCN	0.597863
psb-walk-GIN	0.3638086	pascal-lap-GIN	0.11533	peptides-walk-GatedGCN	0.59613
psb-edge-lap-GCN	0.3600521	pascal-edge-walk-GIN	0.113744	peptides-base-GIN	0.588791
psb-edge-walk-GAT	0.3484621	pascal-edge-GIN	0.107296	peptides-edge-lap-GAT	0.588026
psb-lap-GatedGCN	0.3480012	pascal-edge-lap-GatedGCN	0.105592	peptides-lap-GAT	0.58162
psb-lap-partial-GAT	0.3459151	pascal-base-GIN	0.105413	peptides-edge-walk-GCN	0.577752
psb-edge-lap-GatedGCN	0.344931	pascal-lap-GatedGCN	0.103007	peptides-edges-GCN	0.572586
psb-edge-walk-GatedGCN	0.3344378	pascal-edge-walk-GatedGCN	0.100233	peptides-edge-GIN	0.568688
psb-edge-lap-GAT	0.3320493	pascal-base-GatedGCN	0.099813	peptides-edge-lap-GatedGCN	0.568199
psb-base-GIN	0.3229548	pascal-walk-GatedGCN	0.095798	peptides-edge-GAT	0.567376
psb-edge-lap-partial-GAT	0.3198476	pascal-edge-GatedGCN	0.087692	peptides-base-GAT	0.567336

psb-walk-partial-GAT	0.3050942	pascal-edges-GCN	0.081265	peptides-walk-GCN	0.566612
psb-edge-walk-partial-GAT	0.2983456	pascal-lap-GCN	0.074416	peptides-edge-GatedGCN	0.566249
psb-edge-GIN	0.2618942	pascal-walk-GCN	0.074002	peptides-edge-lap-GCN	0.547736
psb-base-GAT	0.26137	pascal-edge-lap-GCN	0.07195	peptides-base-GatedGCN	0.547136
psb-base-GatedGCN	0.2448564	pascal-edge-walk-GCN	0.071276	peptides-base-GCN	0.54628
psb-edge-GAT	0.2312569	pascal-base-GCN	0.071075	peptides-edge-lap-partial-GAT	0.525215
psb-base-GCN	0.2261042	pascal-walk-partial-GAT	0.04805	peptides-edge-walk-partial-GAT	0.519267
psb-edges-GCN	0.2182385	pascal-partial-GAT	0.047904	peptides-walk-partial-GAT	0.518407
psb-edge-partial-GAT	0.2009716	pascal-edge-lap-partial-GAT	0.047881	peptides-lap-partial-GAT	0.513495
psb-partial-GAT	0.1967177	pascal-lap-partial-GAT	0.04785	peptides-edge-walk-GatedGCN	0.501384
psb-edges-GatedGCN	0.1858331	pascal-edge-walk-partial-GAT	0.047789	peptides-partial-GAT	0.487976
psb-addedges-GatedGCN	0.1858331	pascal-edge-partial-GAT	0.047615	peptides-edge-partial-GAT	0.475301

A1. Full results from traversing different combinations of transformer-based techniques.

REFERENCES

Alon, Uri, and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205,* 2020.

Dwivedi, Vijay Prakash, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *arXiv preprint arXiv:2206.08164,* 2022.

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

Kreuzer, Devin and Beaini, Dominique and Hamilton, William L. and Létourneau, Vincent and Tossou, Prudencio., Rethinking Graph Transformers with Spectral Attention. *arXiv preprint arXiv: 2106.03893*, 2021.

Li, Yujia, et al. "Gated graph sequence neural networks." arXiv preprint arXiv:1511.05493 ,2015.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint cs.LG/1905.10947*, 2019.

Shilane, Philip, et al. "The princeton shape benchmark." Proceedings Shape Modeling Applications, 2004.. IEEE, 2004.

Xu, Keyulu and Hu, Weihua and Leskovec, Jure and Jegelka, Stefanie. How Powerful are Graph Neural Networks? *arXiv preprint arXiv: 1810.00826*, 2018.