Transformers over Directed Acyclic Graphs

Yuankai Luo
Beihang University
luoyk@buaa.edu.cn

Veronika Thost*
MIT-IBM Watson AI Lab, IBM Research
veronika.thost@ibm.com

Lei Shi*
Beihang University
leishi@buaa.edu.cn

1 Introduction

Graph-structured data is ubiquitous in various disciplines [1–3] and hence graph representation learning has the potential to provide huge impact. There are various types of *graph neural networks* (GNNs), the majority of which is based on a so-called message-passing scheme where node representations are computed iteratively by aggregating the embeddings of neighbor nodes [1]. Yet, this mechanism in its basic form has limited expressivity [4] and research is focusing on extensions [5].

Transformer models have recently gained popularity in graph learning as they have the potential to learn complex relationships beyond the ones captured by regular GNNs, and in a different way [6, 7]. Technically, they can be considered as graph neural networks operating on fully-connected computational graphs, decoupled from the input graphs. The main research question in this context is how to inject the structural bias of the given input graphs (i.e., which nodes are actually connected) into the transformer architecture by adapting their attention and positional encoding appropriately. Several promising proposals have been made to encode undirected molecular graphs [8, 9] and recent works take into account the scalability challenge [10], also over larger network graphs [11, 12].

We focus on *directed acyclic graphs* (DAGs), which are of special interest across various domains; examples include parsing results of source code [13], logical formulas [14], conversational emotion recognition [15], as well as probabilistic graphical models and neural architectures [16]. In a DAG, the edges define a partial order over the nodes. This partial order represents an additional strong inductive bias, which offers itself to be integrated into the models.

In fact, various kinds of neural networks tailored to DAG structures have been proposed over the years; from early descriptions of recursive neural networks over DAGs [17, 18], which already point out a similarity to sequence learning, to more recent works, such as DAG-RNN [19], DAG-LSTM [14], D-VAE [16], and DAGNN [20]. The latter models focus on encoding the entire DAG and, in a nutshell, compute that embedding in a message-passing fashion by iterating over the DAG nodes in an asynchronous way, given by the partial order, and thereafter aggregating the final node representations into one for the DAG. This procedure yields state-of-the-art performance, but its asynchronous nature leads to comparatively (to regular GNNs) very slow performance. The more parallel computation of transformers more would seem to make them more suitable models and [21] have recently proposed one such model, PACE, which shows convincing performance in terms of both quality and efficiency.

In this paper, we focus on *transformers over DAGs* more generally, motivated by the highly flexible, parallel, and expressive nature of their computation. In particular, their success in sequence learning opens up the question how they can be tailored to DAGs, which are essentially sequential graphs. Based on the above-mentioned insights in DAG learning, we propose a straightforward and efficient **DAG attention framework**, which *effectively biases any transformer towards DAGs*.

- As outlined in Figure 1, we (1) adapt the attention mechanism by restricting the receptive field of each node to its predecessor and successor nodes so that it faithfully *captures the DAG structure - in terms of its reachability relation* - and at the same time gets *considerably more efficient* than the regular quadratic complexity; and (2) employ the positional encoding in a complementary way, to further bias the computation towards the DAG topology, by explicitly *encoding the node depth*.
- We show that the attention is not restricted effectively, even if it seems so technically. Moreover, we draw a connection to random walk theory.

*Corresponding author.

Y. Luo et al., Transformers over Directed Acyclic Graphs (Extended Abstract). Presented at the Second Learning on Graphs Conference (LoG 2023), Virtual Event, November 27–30, 2023.

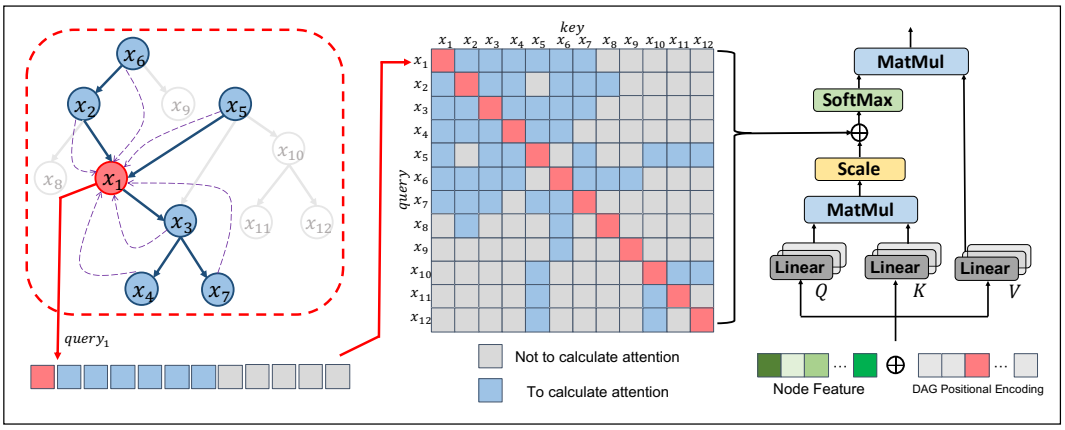

Figure 1: Overview of our DAG attention. A mask matrix is used to restrict the receptive field of the node under consideration to nodes that are directly reachable in the DAG. Our positional encoding (right), additionally captures its position as its depth in the DAG.

- We rigorously evaluate our proposal in ablation studies and show that it successfully improves different kinds of baseline transformers, from vanilla transformers [22] to state-of-the-art graph transformers [11, 23–25], over various types of DAG data. Our experiments range from classifying source code graphs to nodes in citation networks, and go far beyond related works' problem focus. Most importantly, our proposal is proven *effective in two important aspects: in making graph transformers generally outperform graph neural networks tailored to DAGs and in improving SOTA graph transformer performance in terms of both quality and efficiency.*

In this abstract, we give an overview of our main findings; for details and additional results, see the full version of this paper [26]. Our code is available at https://github.com/LUOyk1999/DAGformer.

2 Transformers for Directed Acyclic Graphs

Directed Acyclic Graphs (DAGs). We refer to a *graph* as tuple $G = (V, E, \mathbf{X})$, with node set V, edge set $E \subseteq V \times V$, and node features $\mathbf{X} \in \mathbb{R}^{n \times d}$, with each row representing the feature vector of one node, n denoting the number of nodes, and d the feature dimension. $x_v \in \mathbb{R}^d$ denotes the features of node v. A *directed acyclic graph* is a directed graph without directed cycles. For any DAG G, one can define a unique *strong partial order* $\leqslant$ on the node set V, such that for all pairs of nodes $u, v \in V$, $u \leqslant v$ if and only if there is a directed path from u to v. We define the *reachability relation* $\leqslant$ over a DAG based on that partial order. That is, $u \leqslant v$ if and only if v is reachable from u; further, if $u \leqslant v$, then u is called a *predecessor* of v, and v is a *successor* of u. All nodes without predecessors are *source* nodes, and all nodes without successors are *sink* nodes. The *depth* of a node $v \in V$ is defined as follows. The *depth* of a DAG is the maximum depth over its nodes.

$$depth(v) = \begin{cases} 0 & \text{if } v \text{ is a source node} \\ 1 + \max\limits_{(u,v)\in E} depth(u) & \text{otherwise} \end{cases}$$

Attention based on DAG Reachability (DAGRA). In contrast to regular graphs, the partial order in DAGs creates particular relations between connected nodes, in the sense that a given node's predecessors and successors are most likely more important to it than other graph nodes. Note that this intuition is also captured in the processing of DAG neural networks. Hence the reachability relation suggests itself to be exploited in our architecture. We apply it to bias the receptive field of nodes during attention to their predecessors and successors in the graph, similar to [21]. [22] already mentioned the possibility to use restricted attention in order to reduce complexity and, indeed, our proposal does not only yield an architecture which is biased towards the DAG structure, but additionally a considerably more efficient model. We compute our *reachability-based attention* as:

$$\text{Attention}_{\text{DAGRA}}(x_v) = \sum_{u \in N(v)} \frac{\kappa(x_v, x_u)}{\sum_{w \in N(v)} \kappa(x_v, x_w)} f(x_u), \forall v \in V, \tag{1}$$

$$\kappa(x, x') = \exp\left(\frac{\langle x\mathbf{W_Q}, x'\mathbf{W_K}\rangle}{\sqrt{d_K}}\right), \tag{2}$$

where $N(v) = \{(u, v) \in \leqslant\} \cup \{(v, u) \in \leqslant\}$ represents the receptive field of node v. The input node features $\mathbf{X}$ are projected as usual by three matrices $\mathrm{W}_{\mathbf{Q}} \in \mathbb{R}^{d \times d_K}$, $\mathrm{W}_{\mathbf{K}} \in \mathbb{R}^{d \times d_K}$ and $\mathrm{W}_{\mathbf{V}} \in \mathbb{R}^{d \times d_K}$ to the corresponding representations of query $\mathbf{Q}$, key $\mathbf{K}$, and value $\mathbf{V}$, $f(x) = \mathbf{W}_{\mathbf{V}} x$, and κ is a non-symmetric exponential kernel. Note that our formulation is based on a reformulation of the original self-attention mechanism as a kernel smoother as proposed in [24].

DAG Positional Encodings based on DAG Depth (DAGPE). Positional encodings have been recognized as important and proven effective for incorporating graph structure into transformers. Observe that the sequential nature of DAGs makes them possess special position information, the depth of a node within the DAG. We propose to include this knowledge in the form of *DAG positional encodings* as follows, exactly as suggested for the original transformer architecture [22]:

$$PE_{(v,2i)} = \sin\left(\frac{depth(v)}{10000^{\frac{2i}{d}}}\right), \quad PE_{(v,2i+1)} = \cos\left(\frac{depth(v)}{10000^{\frac{2i}{d}}}\right),$$

where d is the node feature dimension and i the index of the dimension under consideration.

DAG Attention. We obtain the following model, combining DAGRA and DAGPE, for $v \in V$:

$$\mathrm{Attention}_{\mathrm{DAG}}\left(x_v\right) = \sum_{u \in N(v)} \frac{\kappa\left(x_v + PE_v, x_u + PE_u\right)}{\sum_{w \in N(v)} \kappa\left(x_v + PE_v, x_w + PE_w\right)} f\left(x_u\right). \tag{3}$$

Our attention incorporates both similarity of node features and of node depths. We argue that these are the most critical aspects of DAGs and our evaluation will show their impact and general effectiveness. In particular, note that most kinds of DAG data (e.g., citation networks) do not require us distinguishing between predecessor or successor nodes of same depth.

Theoretical Intuition. While we technically restrict the attention to reachable nodes, we can show that this design offers similar expressivity to regular transformers. Specifically, each node directly communicates with at least one source node by the DAG structure. Hence, 2 transformer layers are always enough to establish communication beyond two nodes that have a common source node. We can generalize this result. Hence, our architecture's bias emphasizes DAG relationships while re-directing the remaining relationships in the regular transformer's full attention matrix. This can also be shown to be in line with random walk theory.

Implementation. Our DAG attention framework can be implemented in two ways, either using a mask matrix on top of existing transformer models, or based on message-passing GNNs. Given usually sparse graphs, observe that the latter yields considerably performance gains with a complexity of $O(|V| \times \overline{deg}^{l} \times d)$, with average node degree $\overline{deg}$ and the DAG depth l.

3 Evaluation

Datasets. We conducted experiments over a wide range of data and tasks including **ogbg-code2** [27], a large dataset of ASTs derived from Python methods; **NA** [16], a dataset with much smaller graphs, containing neural architecture DAGs; **Self-citation** [28, 29], containing scholars' academic self-citation networks; and the established **Cora, Citeseer, Pubmed** [30] where, only for our method, we removed a small number of cyclic citation links to make them DAGs.

Baselines. We chose **MixHop** [31], **LDS-GNN** [32], **IDGL** [33] and **PNA**, as more recent GNN proposals. In terms of transformer models, we considered vanilla **Transformer (TF)**, **Graph Transformer (GT)**, **GraphTrans**, **SAT**, and the recent and competitive **GraphGPS** and **NodeFormer**. Lastly, we consider neural networks tailored to DAGs: **D-VAE**, **DAGNN**, and **PACE**.

DAG+ Models. We implemented our DAG attention on top of vanilla Transformer, GraphTrans, SAT, GraphGPS and NodeFormer *only (1) modifying their self-attention module by restricting attention in terms of reachability and (2) using DAGPE instead of the original one.* For fair comparisons, we *use the same hyperparameters* and readout as baseline transformers.

Overall Performance, Tables 1, 2, 3 and 4. First, observe that the results are very consistent, although the datasets differ greatly in size, DAG sizes, DAG shape (e.g., in ogbg-code2 we have trees), and nature of data (e.g., node features). The message-passing GNNs represent standardized baselines but do not reach the performance of networks tailored to DAGs, such as DAGNN. Note that the latter is however comparatively bad on the node-level task. This can be explained by its processing.

Table 1: Code graph classification on ogbg-code2. The baseline results were taken from the OGB leaderboard.

Model	Valid F1 (%)	Test F1 (%)	Time(epoch)
PNA	14.5 ± 0.3	15.7 ± 0.3	427s
DAGNN	16.1 ± 0.4	17.5 ± 0.5	6018s
PACE	16.3 ± 0.3	17.8 ± 0.2	2410s
Transformer	15.5 ± 0.2	16.7 ± 0.2	1817s
DAG+Transformer	**17.4** ± 0.1	**18.8** ± 0.2	**591s**
GraphTrans	16.6 ± 0.1	18.3 ± 0.2	1117s
DAG+GraphTrans	**17.0** ± 0.2	**18.7** ± 0.2	**526s**
GraphGPS	17.4 ± 0.2	18.9 ± 0.2	1919s
DAG+GraphGPS	**17.6** ± 0.1	**19.3** ± 0.2	**608s**
SAT (SOTA)	17.7 ± 0.2	19.4 ± 0.3	2437s
DAG+SAT	**18.5** ± 0.1	**20.2** ± 0.2	**681s**

Table 2: Node classification results for the self-citation dataset; AP (%) and ROC-AUC (%).

Model	AP ↑	ROC-AUC ↑
PNA	62.4 ± 0.7	81.0 ± 0.4
DAGNN	61.2 ± 0.6	81.0 ± 0.3
PACE	52.1 ± 1.8	75.9 ± 0.7
Transformer	56.8 ± 1.8	78.7 ± 0.3
DAG+Transformer	**63.8** ± 0.8	**82.2** ± 0.5
GraphGPS	61.6 ± 2.6	**81.3** ± 0.6
DAG+GraphGPS	**63.5** ± 1.2	80.8 ± 0.5
SAT	59.8 ± 1.7	79.8 ± 0.7
DAG+SAT	**62.7** ± 1.5	**80.6** ± 0.7
NodeFormer	39.6 ± 0.6	69.4 ± 0.3
DAG+NodeFormer	**64.9** ± 0.8	**81.7** ± 0.8

Table 3: Node classification accuracy (%).The baseline results were taken from [25].

Model	Cora	Citeseer	Pubmed
MixHop	87.59 ± 0.52	73.64 ± 0.73	89.32 ± 0.25
IDGL	87.88 ± 0.34	74.32 ± 0.51	89.22 ± 0.14
LDS-GNN	87.82 ± 0.62	75.22 ± 0.23	OOM
PACE	79.47 ± 0.63	73.65 ± 1.23	OOM
Transformer	75.92 ± 0.86	72.23 ± 1.06	OOM
DAG+Transformer	**87.80** ± 0.53	**74.42** ± 0.22	**89.01** ± 0.13
SAT	75.18 ± 0.62	74.88 ± 0.73	OOM
DAG+SAT	**87.48** ± 0.37	**76.64** ± 0.26	**89.17** ± 0.15
NodeFormer	88.80 ± 0.26	76.33 ± 0.59	89.32 ± 0.25
DAG+NodeFormer	**90.49** ± 0.17	**78.24** ± 0.33	**89.44** ± 0.24

Table 4: Regression. Predictive performance of latent representations over NA.

Model	RMSE ↓	Pearson's r ↑
D-VAE	0.375 ± 0.003	0.924 ± 0.001
DAGNN	0.264 ± 0.004	0.964 ± 0.001
PACE	0.254 ± 0.002	0.964 ± 0.001
Transformer	0.285 ± 0.004	0.957 ± 0.001
GT	0.275 ± 0.003	0.961 ± 0.001
DAG+Transformer	**0.253** ± 0.002	**0.966** ± 0.001
GraphGPS	0.306 ± 0.004	0.950 ± 0.001
DAG+GraphGPS	**0.267** ± 0.005	**0.964** ± 0.001
SAT	0.298 ± 0.003	0.952 ± 0.001
DAG+SAT	**0.262** ± 0.004	**0.964** ± 0.001

It computes node representations in the order of the partial order, and hence the representations of nodes in the beginning of the order contain only few information about the entire graph. Intuitively, transformers should be able to capture this information, but the transformers we tested do neither perform better, not even the ones tailored to graphs. This shows that they are missing information captured by those message-passing neural networks that were tailored to DAGs. Yet, the results clearly show that our DAG attention is successful in providing good improvements, over the best graph transformer SAT and even better ones over vanilla Transformer. On ogbg-code2, the improvement is smaller for GraphTrans and SAT. However, this benchmark task is heavily dependent on other features (e.g., language understanding) and hence presents a special challenge; in this regard, the NA and Self-citation contain "cleaner" graphs. It is interesting to see that our framework lifts the transformers from below the level of DAGNN to above, in terms of all metrics, over these two datasets. Lastly, we observe that the PACE transformer tailored to DAGs is similarly outperformed by our simpler, but more effective technique. In summary, our DAG attention makes the transformers outperform (1) the original transformers and (2) the neural networks tailored to DAGs. This shows that the straightforward, DAG-specific bias provided by DAG attention is the right bias in many scenarios.

Ablation Studies and Analysis. (1) Our ablation studies removing DAGRA and DAGPE individually confirm the impact of both modules. (2) We also experimented with replacing DAGPE with LapPE [6] and the random-walk-based RWPE [34] to show that the DAG-specific nature of our PE is of advantage. (3) We experimented with adding attention bias which captures the graph structure more directly (e.g., shortest-path [8] and edge directionality), although they are implicit in DAGPE; interestingly, the more direct representation does not make a noticeable difference.

4 Conclusions

Based on the insights from models for directed acyclic graphs, we have developed a framework to bias transformer models towards DAGs. It allows for incorporating the main characteristic of DAGs - their partial order - into any transformer architecture, by encoding the reachability relation and positions resulting from it. Our architecture is simple and universal as we have demonstrated in our experiments. Most importantly, it has been proven effective in improving existing graph transformers over DAGs with improvements in quality and impressive increases in efficiency.

References

[1] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017. 1

[2] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinform.*, 34(13):i457–i466, 2018.

[3] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. *CoRR*, abs/2002.09405, 2020. URL https://arxiv.org/abs/2002.09405. 1

[4] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018. 1

[5] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020. 1

[6] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020. 1, 4

[7] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021. 1

[8] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021. 1, 4, 12

[9] Jerret Ross, Brian Belgodere, Vijil Chenthamarakshan, Inkit Padhi, Youssef Mroueh, and Payel Das. Molformer: Large scale chemical language representations capture molecular structure and properties. 2022. 1

[10] Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *arXiv preprint arXiv:2206.08164*, 2022. 1

[11] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *arXiv preprint arXiv:2205.12454*, 2022. 1, 2

[12] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. Nagphormer: Neighborhood aggregation graph transformer for node classification in large graphs. *arXiv preprint arXiv:2206.04910*, 2022. 1

[13] Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):1–37, 2018. 1, 9

[14] Maxwell Crouse, Ibrahim Abdelaziz, Cristina Cornelio, Veronika Thost, Lingfei Wu, Kenneth Forbus, and Achille Fokoue. Improving graph neural network representations of logical formulae with subgraph pooling. *arXiv preprint arXiv:1911.06904*, 2019. 1

[15] Weizhou Shen, Siyue Wu, Yunyi Yang, and Xiaojun Quan. Directed acyclic graph network for conversational emotion recognition. *arXiv preprint arXiv:2105.12907*, 2021. 1

[16] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. *Advances in Neural Information Processing Systems*, 32, 2019. 1, 3, 8, 10, 11

[17] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997. 1

[18] Paolo Frasconi, Marco Gori, and Alessandro Sperduti. A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks*, 9(5):768–786, 1998. 1

[19] Bing Shuai, Zhen Zuo, Bing Wang, and Gang Wang. Dag-recurrent neural networks for scene labeling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3620–3629, 2016. 1

[20] Veronika Thost and Jie Chen. Directed acyclic graph neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=JbuYF437WB6. 1, 11

[21] Zehao Dong, Muhan Zhang, Fuhai Li, and Yixin Chen. Pace: A parallelizable computation encoder for directed acyclic graphs. In *International Conference on Machine Learning*, pages 5360–5377. PMLR, 2022. 1, 2

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 2, 3

[23] Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems*, 34:13266–13279, 2021. 2

[24] Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, pages 3469–3489. PMLR, 2022. 3, 11

[25] Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in Neural Information Processing Systems*, 35:27387–27401, 2022. 2, 4, 11, 12

[26] Yuankai Luo, Veronika Thost, and Lei Shi. Transformers over directed acyclic graphs. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=g49s1N5nmO. 2

[27] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020. 3, 8, 10

[28] The acl anthology reference corpus. https://aclanthology.org/, Nov. 2021. 3, 11

[29] Yuankai Luo, Lei Shi, Mufan Xu, Yuwen Ji, Fengli Xiao, Chunming Hu, and Zhiguang Shan. Impact-oriented contextual scholar profiling using self-citation graphs. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 4572–4583, 2023. 3

[30] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008. 3, 10, 11

[31] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019. 3

[32] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *International conference on machine learning*, pages 1972–1982. PMLR, 2019. 3

[33] Yu Chen, Lingfei Wu, and Mohammed Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *Advances in neural information processing systems*, 33:19314–19326, 2020. 3

[34] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2021. 4

[35] Sergey Brin. The pagerank citation ranking: bringing order to the web. *Proceedings of ASIS, 1998*, 98:161–172, 1998. 8

[36] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*. 8

[37] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019. 9, 10

[38] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018. 10, 11

[39] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 11

[40] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18, 2005. 11

[41] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 12

[42] Qitian Wu, Chenxiao Yang, Wentao Zhao, Yixuan He, David Wipf, and Junchi Yan. DIFFormer: Scalable (graph) transformers induced by energy constrained diffusion. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=j6zUzrapY3L. 13

[43] Scott Freitas and Yuxiao Dong. A large-scale database for graph representation learning. *Advances in neural information processing systems*, 2021. 13

A DAG Attention

Bounded Reachability. While graphs to classify are usually of manageable size, graph learning in general may face much larger ones. For this reason, we formulate our model in a more general way, based on a bounded reachability relation, representing the receptive field of each node: $N_k(v) = \{(u, v) \in \leqslant_k\} \cup \{(v, u) \in \leqslant_k\}$. Note that we generally used the maximal $k = \infty$ in our evaluation, and therefore only consider $N = N_\infty$ in the main text.

A.1 Expressive Power of DAG Attention

Technically, we restrict the attention to reachable nodes and, in this way, obtain considerable efficiency gains. Yet, our architecture is tailored to the special DAG structure, and we can show that this design offers similar expressivity to regular transformers.

It is important to note that in our framework, all nodes directly communicate with at least one source node (i.e., node without predecessors) by the DAG structure. This is specifically the case because we do not restrict the radius k of the receptive field (not introduced in the main text), but consider $k = \infty$. Hence, 2 layers are always enough to establish communication beyond any two nodes that have a common source node. Furthermore, especially DAG classification datasets usually contain DAGs with a single source node (e.g., ogbg-code2 [27] and NA [16]).

For DAGs with m source nodes we need $2m$ layers for full communication, if we assume the DAG to be connected. In the latter case, every pair of source nodes has a common successor through which communication can happen. Further, connectedness is a reasonable assumption, otherwise communication is likely not needed in most scenarios.

The source nodes may seem to represent a certain kind of bottlenecks since, essentially, our architecture's bias emphasizes DAG relationships while re-directing the remaining relationships in the regular Transformer's full attention matrix. But in the next section, we show that the importance of relationships we model is in line with well-known random walk theory, and our qualitative performance increases also demonstrate that putting more emphasis on DAG relationships can be beneficial in a variety of use cases.

A.2 Further Theoretical Intuition of DAG Attention

To justify our attention method, focusing only on reachable nodes, we draw a connection to PageRank [35, 36]. Specifically, we consider a PageRank variant that takes the root node into account - personalized PageRank. We define the root node x via the teleport vector i_x, which is a one-hot indicator vector. The personalized PageRank can be obtained for node x using the recurrent equation $\pi_G(i_x) = (1 - \alpha)A_{rw}\pi_G(i_x) + \alpha i_x$, with teleport (or restart) probability $\alpha \in (0, 1]$. Solving this equation, we obtain: $\pi_G(i_x) = \alpha(I_n - (1 - \alpha)A_{rw})^{-1}i_x$, where $A_{rw} = AD^{-1}$, with A and D being the adjacency and the degree matrix, respectively [36].

We invert the directions of the edges in G to create a reverse DAG $\tilde{G}$. We can show that, for every node x, only nodes y not reachable from x (i.e., $y \notin N_\infty(x)$) will satisfy that $\pi_G(i_x)[y] + \pi_{\tilde{G}}(i_x)[y]$ (the y-th element of $\pi_{\tilde{G}}(i_x)$) equals 0.

Proof. Personalized PageRank is defined as $\pi_G(i_x) = (1 - \alpha)A_{rw}\pi_G(i_x) + \alpha i_x$. Through rearrangement, we obtain: $(I_n - (1 - \alpha)A_{rw})\pi_G(i_x) = \alpha i_x$. Now we prove that $(I_n - (1 - \alpha)A_{rw})$ is an invertible matrix. The matrix is invertible iff the determinant $\det(I_n - (1 - \alpha)A_{rw}) \neq 0$, which is the case iff $\det(A_{rw} - \frac{1}{1-\alpha}I_n) \neq 0$, i.e., iff $\frac{1}{1-\alpha}$ is not an eigenvalue of A_{rw}. This value is always larger than 1 since the teleport probability $\alpha \in (0, 1]$. However, the largest eigenvalue of the row-stochastic matrix A_{rw} is 1, as can be proven using the Gershgorin circle theorem. Hence, $\frac{1}{1-\alpha}$ cannot be an eigenvalue and $(I_n - (1 - \alpha)A_{rw})$ is an invertible matrix. So we obtain $\pi_G(i_x) = \alpha(I_n - (1 - \alpha)A_{rw})^{-1}i_x$. Because the spectral radius (largest eigenvalue) of matrix $(1 - \alpha)A_{rw}$ is less than 1, from the Neumann series theorem, we obtain

$$(I_n - (1 - \alpha)A_{rw})^{-1} = \sum_{m=0}^{\infty} \left((1 - \alpha)\, A_{rw}\right)^m = \sum_{m=0}^{\infty} \left((1 - \alpha)\, AD^{-1}\right)^m.$$

Because D is a diagonal matrix and $1 - \alpha$ is a constant, they do not affect whether an element in the matrix is 0 or not. Therefore, we only need to discuss $\sum_{m=0}^{\infty} A^m$. Because A is the adjacency

matrix of the DAG G, in this matrix, iff there is not a directed path from x to y ($x \not\leqslant y$), we deduce that $(\sum_{m=0}^{\infty} A^m)[x, y] = 0$. This leads us to the conclusion that $\pi_G(i_x)[y] = 0$.

We invert the directions of the edges in G to create a reverse DAG $\tilde{G}$. We can also obtain that, iff $x \not\leqslant y$, then $\pi_{\tilde{G}}(i_x)[y] = 0$.

Therefore, for every node x, only nodes y that $y \notin N_\infty(x)$ will satisfy that $\pi_G(i_x)[y] + \pi_{\tilde{G}}(i_x)[y]$ equals 0. □

This means that for the random walk's limit distribution, the probability of nodes that are not reachable from x is 0.

A.3 Implementation

We describe two ways of implementing our model, especially DAGRA, based upon transformers and message-passing GNNs, respectively.

Masked Attention for Transformers. As shown in Figure 1, we can implement DAG attention in a very straightforward fashion using a mask that masks out node pairs based on the DAG reachability relation as follows, with the attention mask $\mathbf{M}$ being defined as a symmetric matrix over node pairs $(v, u) \in V \times V$.

$$\text{Attention}\,(\mathbf{X}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{\mathrm{d}_K}} + \mathbf{M}\right)\mathbf{V}, \quad \mathbf{M}(v,u) = \begin{cases} 0 & \text{if } u \in N_k(v) \\ -\infty & \text{otherwise.} \end{cases}$$

While this masking represents a simple technique to extend and bias existing transformer models to DAGs, the resulting architecture does not benefit from the restricted node set to be considered, leading to unnecessary, costly matrix operations during attention calculation. Moreover, it consumes $O(|V|^2)$ of additional memory to store $\mathbf{M}$. Thus, the runtime complexity per layer is still $O(|V|^2 d)$ for the vanilla transformer - and may be even higher, depending on the underlying transformer.

DAG Attention using Message Passing. Based on the formulation in Equation (3), we propose to follow the message-passing scheme; that is, for a node v, we compute $N_k(v)$ and only aggregate messages from the nodes in that set to compute the DAG attention. For the latter aggregation, we can use readily available frameworks, such as PyG [37]. We analyze the complexity of this proposal below.

A.4 Computational Complexity

Clearly, the time complexity of the proposed model is lower than that of the standard transformer. We consider two computation steps:

Computing DAG Reachability. To obtain N_k, we compute the transitive closure of E for each node in the graph using a breadth-first search starting at the node and iteratively expanding N_k based on E. Hence, the overall complexity of this step is $O(|E||V|)$. Observe that, during training, we can consider this step as pre-processing since it only has to be run once, in the very beginning.

DAG Attention. The matrix product $\mathbf{Q}\mathbf{K}^T$ of self-attention has a cost $O(|V|^2 d)$ which is quadratic in the number of nodes in G, and hence especially critical for large graphs. Yet, for our DAG attention, we only aggregate messages from reachable nodes. Therefore, the time complexity of reachability relation attention is $O(|V| \times n_k \times d)$, where n_k is the average size of N_k. In the worst case, we have $n_k = |V| - 1$, but we assume that $n_k << |V|$ in general. Especially for sparse graphs, the complexity gets significantly lower, i.e., $O(|V| \times \overline{deg}^k \times d)$, where $\overline{deg}$ is the average node degree.

Finally, observe that directed rooted trees represent a special kind of DAGs broadly seen across domains, such as abstract syntax trees of source code [13]. For them, the runtime of DAG attention scales almost linearly, as illustrated in the following theorem.

Theorem 1. *In a directed rooted tree $T = (V, E)$ with the number of nodes $|V|$, the runtime of DAG attention is $O(|V| \times k \times \Delta^+ \times d)$, where Δ^+ is the maximum out degree of the T and d is feature dimension. When $k = \infty$, the runtime of DAG attention is $O(|V| \times \ell \times \Delta^+ \times d)$, where ℓ is the depth of T.*

Table 5: Statistics of the datasets we used.

	ogbg-code2	NA	Self-citation	Cora	Citeseer	Pubmed
# graphs	452,741	19,020	1,000	1	1	1
Avg # nodes	125.2	8.0	59.1	2,708	3,327	19717
Avg n_∞	9.78	7.00	4.30	20.88	5.33	60.56

Proof. In DAG attention, for each node v, we need to calculate the attention to $N_k(v)$, whose runtime is upper bounded by $O(|N_k(v)| \times d)$. So the runtime of DAG attention is linear in the sum of the sizes of receptive fields $\sum_{v\in V} |N_k(v)|$. We only need to show that $\sum_{v\in V} |N_k(v)| \leqslant |V| \times k \times \Delta^+$. When $k = \infty$, $\sum_{v\in V} |N_k(v)| \leqslant |V| \times \ell \times \Delta^+$.

Next, we prove this theorem for the case of $k = \infty$. For bounded k, the proof is similar. We prove by induction on the depth ℓ of T.

We assume that $\Delta^+ \geqslant 2$, since when $\Delta^+ = 1$, T is a 1-ary tree, $\sum_{v\in V} |N_\infty(v)| = |V| \times |V| \leqslant |V| \times \ell \times \Delta^+$.

For the basis step, when $\ell = 0$, T only has one node. So $\sum_{v\in V} |N_\infty(v)| = 0 \leqslant |V| \times \ell \times \Delta^+$.

For the inductive step, we assume that
$\sum_{v\in V} |N_\infty(v)| \leqslant |V| \times \ell \times \Delta^+$ holds when $\ell \geqslant 0$. Consider a tree T_{new} of depth $\ell + 1$ whose root is denoted by r. Then the sub-trees whose roots are the children of r have depth at most ℓ. Let s be the number of these sub-trees, $s \leqslant \Delta^+$, and V_i be the node set of the i-th sub-tree. Since the number of reachability relations with r involved is $2(|V| - 1)$, we have

$$
\begin{aligned}
& \sum_{v\in V} |N_\infty(v)| \\
= & \sum_{i\in[s]} \sum_{v\in V_i} |N_\infty(v)| + 2(|V| - 1) \\
\leqslant & |V_1| \times \ell \times \Delta^+ + ... + |V_s| \times \ell \times \Delta^+ + 2(|V| - 1) \\
\leqslant & \ell \times \Delta^+ \times (|V| - 1) + 2(|V| - 1) \\
\leqslant & \ell \times \Delta^+ \times (|V| - 1) + \Delta^+ \times (|V| - 1) \\
\leqslant & (\ell + 1) \times \Delta^+ \times |V|
\end{aligned}
$$

This completes the proof of Theorem 1 for the case of $k = \infty$. □

B Experimental Details

B.1 Computing Environment

Our implementation is based on PyG [37]. All reported results are averaged over 10 runs using different random seeds. The experiments are conducted with two RTX 3090 GPUs.

B.2 Dataset Details

We provide details of the datasets used in our experiments, including ogbg-code2 [27], Neural architectures (NA) [16], Self-citation, Cora, Citeseer and Pubmed [30]. Table 5 shows the diversity of the datasets we used.

ogbg-code2. ogbg-code2 [27] is a dataset containing 452,741 Python method definitions extracted from thousands of popular Github repositories. It is made up of Abstract Syntax Trees as DAGs where the task is to correctly classify the sub-tokens that comprise the method name. The min/avg/max numbers of nodes in the graphs are 11/125/36123, respectively. We use the standard evaluation metric (F1 score) and splits provided by [27].

Neural architectures (NA). This dataset is created in the context of neural architecture search. It contains 19,020 neural architectures generated from the ENAS software [38]. Each neural architecture has 6 layers (i.e., nodes) sampled from 6 different types of components, plus an input and output

Table 6: The statistics of academic self-citation dataset.

	Min	Avg	Max
# nodes	30	59	296
DAG depth	1	5	21
Node degree	0	1.8	83

layer. The input node vectors are one-hot encodings of the component types. The weight-sharing accuracy [38] (a proxy of the true accuracy) on CIFAR-10 [39] is taken as performance measure. Since it is a regression task, the metrics are RMSE and Pearson's r. We use the splits as is used in [20]. Details about the generation process can be found in [16].

Self-citation. The academic self-citation dataset is a directed acyclic graph, representing the scholar's academic self-citation network on the natural language processing (NLP) field extracted from ACL Anthology Reference Corpus [28]. This dataset contains 1000 academic self-citation networks from scholars ranked top-1000 by number of papers. In a self-citation networks, each node is a paper of this scholar and each directed edge indicates that one paper is cited by another one. Each graph node includes two quantitative attributes: the publication year, the paper's total citation count. The task is to predict missing citation counts given existing citation counts. Specifically, for each scholar self-citation network, we randomly select 10% of its papers and drop their citation counts, and the node-level task is to predict whether a paper which misses its citation count is highly-cited or not. In Computer Science, Engineering, and Mathematics, for papers in ISI (Web of science database) listed journals that are 10 years old, around 20 citations gets in the top 10%. So we consider papers with a citation count greater than or equal to 20 as highly-cited papers. As the class balance is skewed (only 27.2% of data is positive), we use the Average Precision (AP) and ROC-AUC as the evaluation metrics. We randomly split the dataset into 80% training, 10% valid and 10% test sets. The statistics is shown in Table 6.

Cora, Citeseer and Pubmed [30]. These three datasets are commonly used citation networks for evaluating models on node classification tasks. These datasets are medium-scale networks (with 2K~20K nodes) and the goal is to classify the topics of documents (instances) based on input features of each instance (bag-of-words representation of documents) and graph structure (citation links). For the baselines, the citation links are treated as undirected edges [25] and each document has a class label. Only for our method, we treated citation links as directed edges and removed a small number of cyclic citation links to make them DAGs. We use the same evaluation metric (classification accuracy) and splits provided by NodeFormer [25].

B.3 Hyperparameter and Reproducibility

ogbg-code2. We implemented DAG attention on vanilla Transformer, GraphTrans, GraphGPS and SAT that we only modify the self-attention module to DAG attention module. And we do not use any PE as well as baselines because it makes very little performance improvement [24]. For fair comparisons, we use the same hyperparameters (including training schedule, optimizer, number of layers, batch size, hidden dimension etc.) as baseline models for all of our four versions.

NA. We train the transformer-based baselines (vanilla Transformer, GraphGPS and SAT) in front of DAGNN [20]. For SAT, we use the 3-subtree GNN extractor (GIN). For GraphGPS, we use GatedGCN as GPS-MPNN and vanilla Transformer as GPS-GlobAttn. And we implement DAG attention on those respectively. We follow the exact training settings of DAGNN [20]. Then we also train a sparse Gaussian process (SGP) [40] with a learning rate of 4e-4, a epochs number 200 as the predictive model to evaluate the performance of the latent representations. The transformer-based hyperparameters are summarized in Table 7. All other hyperparameters are the same as those of DAGNN [20].

Self-citation. We implement DAG attention on the vanilla Transformer, GraphGPS, SAT and NodeFormer. In general, we perform a hyperparameter search to produce the results for our model and baselines. The hyperparameters on the academic self-citation dataset are summarized in Table 7, where # Attention heads is tuned for transformer-based models. For SAT, we conduct positional encodings search on LapPE-8, RWPE-20 or None, and we use the 3-subtree GNN extractor (GIN). For GraphGPS, we use GatedGCN as GPS-MPNN and vanilla Transformer as GPS-GlobAttn. The other hyper-parameters for NodeFormer are the default parameters in their public code. In all experiments, we use the validation set to select the best hyperparameters. All our models and baselines are trained

Table 7: Hyperparameter search on different datasets.

Hyperparameter	NA	Self-citation	Cora, Citeseer and Pubmed
# Layers	{2,3}	{2, 3, 4, 5}	{2, 3, 4, 5}
Hidden dimensions	{32,128}	{32, 64, 128}	{16, 32, 64, 128}
Dropout	0.2	{0.1, 0.2, 0.5}	0.0
Learning rate	1e-3	{1e-4, 5-e4, 1e-3, 5e-3}	{1e-3, 1e-2}
# Epochs	100	{50, 100}	1000
Weight decay	None	1e-6	5e-3
# Attention heads	{2, 4}	{2, 4, 8}	{2, 4}

Table 8: Number of parameters for DAG+ models.

Model	ogbg-code2	NA	Self-citation	Cora	Citeseer	Pubmed
DAG+Transformer	127,06k	399k	478k	291k	404k	14k
DAG+SAT	149,53k	631k	710k	276k	165k	14k

with the AdamW optimizer [41] and we use the cross-entropy loss on this classification task. The learning rate scheduler is the cosine scheduler [41].

Cora, Citeseer and Pubmed. We implement DAG attention on the vanilla Transformer, SAT and NodeFormer. And we perform a hyperparameter search to produce the results for our model and those transformer-based baselines in Table 7. We follow all other training setting and hyperparameters of NodeFormer [25]. For SAT, we use the 1-subtree GNN extractor (GCN).

In Table 8, we report the number of parameters for DAG+ models using the hyperparameters selected from Table 7.

B.4 Ablation Study

Table 9: Ablation study results.

Ablation	ogbg-code2		NA		Self-citation	
	Valid F1	Test F1	RMSE ↓	Pearson's r ↑	AP ↑	ROC-AUC ↑
DAG+TF	0.1731 ± 0.0014	0.1895 ± 0.0014	**0.253** ± 0.002	**0.966** ± 0.001	**0.638** ± 0.008	**0.822** ± 0.005
(-) DAGRA	0.1546 ± 0.0018	0.1670 ± 0.0015	0.284 ± 0.003	0.957 ± 0.001	0.573 ± 0.011	0.790 ± 0.003
(-) DAGPE	**0.1739** ± 0.0013	**0.1879** ± 0.0015	0.263 ± 0.002	0.963 ± 0.001	0.594 ± 0.028	0.782 ± 0.018
(+) RWPE	-	-	0.267 ± 0.003	0.962 ± 0.001	0.628 ± 0.014	0.819 ± 0.010
(+) LapPE	-	-	0.271 ± 0.002	0.961 ± 0.001	0.609 ± 0.017	0.786 ± 0.015
(+) SPD [8]	0.1749 ± 0.0011	0.1881 ± 0.0017	-	-	0.639 ± 0.006	0.823 ± 0.004
(+) Edge Direction	0.1751 ± 0.0018	0.1870 ± 0.0021	-	-	0.636 ± 0.015	0.817 ± 0.005
TF	0.1546 ± 0.0018	0.1670 ± 0.0015	0.285 ± 0.004	0.957 ± 0.001	0.568 ± 0.018	0.787 ± 0.003
DAG+SAT	0.1821 ± 0.0013	0.1982 ± 0.0010	**0.262** ± 0.004	**0.964** ± 0.001	**0.627** ± 0.015	**0.806** ± 0.007
(-) DAGRA	0.1773 ± 0.0023	0.1937 ± 0.0028	0.292 ± 0.003	0.954 ± 0.001	0.598 ± 0.031	0.800 ± 0.012
(-) DAGPE	**0.1846** ± 0.0010	**0.2018** ± 0.0021	0.282 ± 0.002	0.958 ± 0.001	0.623 ± 0.014	0.806 ± 0.005
(+) SPD [8]	0.1851 ± 0.0008	0.1991 ± 0.0018	-	-	0.627 ± 0.016	0.810 ± 0.006
(+) Edge Direction	0.1839 ± 0.0014	0.1978 ± 0.0028	-	-	0.623 ± 0.013	0.804 ± 0.007
SAT	0.1773 ± 0.0023	0.1937 ± 0.0028	0.298 ± 0.003	0.952 ± 0.001	0.598 ± 0.017	0.798 ± 0.007

B.5 Additional Results

B.5.1 Semi-supervised Node Classification

We test our model on three citation networks Cora, Citeseer and Pubmed. We implement DAG attention on DIFFormer-a [42]. Table 10 reports the testing accuracy. This show that our framework likely increases both quality and runtime here as well.

Table 10: Node classification accuracy (%).The baseline results were taken from [42].

Model	Cora		Citeseer		Pubmed	
	Accuracy	Time (s)	Accuracy	Time (s)	Accuracy	Time (s)
NodeFormer	83.4 ± 0.2	-	73.0 ± 0.3	-	81.5 ± 0.4	-
DIFFORMER-a	84.1 ± 0.6	14.21	75.7 ± 0.3	8.48	80.5 ± 1.2	1013.31
DAG + DIFFORMER-a	85.1 ± 0.7	10.96	76.2 ± 0.4	7.01	81.6 ± 0.5	84.15

B.5.2 MalNet-Tiny

Here we consider MalNet-Tiny. MalNet-Tiny [43] is a subset of MalNet which consists of function call graphs (FCGs) obtained from Android APKs. This subset includes 5,000 graphs with a maximum of 5,000 nodes, originating from benign software or 4 malware types. The FCGs do not have any original node or edge features. The benchmark version of this dataset usually employs Local Degree Profile as the set of node features. And the objective is to predict the type of software solely based on the structure.

Note that we did not do any hyperparameter tuning. Further, we implemented DAG attention on top of GraphGPS by only modifying the self-attention module, switching it to DAG attention. As shown in Table 11, Our DAG+GraphGPS achieves a test accuracy of 93.45% while only requiring 20 seconds per epoch. This demonstrates the quality and efficiency of DAG attention.

Table 11: Graph classification on MalNet-Tiny.

Model	Accuracy (%)	Time(epoch)
GraphGPS	92.64 ± 0.78	46s
DAG+GraphGPS	93.45 ± 0.41	20s