Efficient Edge-aware Attention Network for Graph Generation

Vincent Jung [*1 2] **Alba Carballo-Castro** [*1] **Yiming Qin** [*1] **Lonneke van der Plas** [2] **Pascal Frossard** [1]

Abstract

Graph generative models commonly use graph-specific Transformer architectures to jointly update node and edge features. However, these architectures do not directly benefit from recent hardware-aware attention implementations, limiting their scalability. This adaptation is non-trivial: edge and graph-level features must be injected into pairwise attention scores, graph masks must be respected, and dynamic edge updates remain a major source of quadratic memory usage. We propose an edge-aware attention layer for graph generation that incorporates structural information through head-wise score modulation, while updating edge representations with a lightweight residual mechanism that retains key attention-score dependent edge updates. This layer can be directly implemented using PyTorch's `FlexAttention`. We achieve similar generation quality as our baseline while reducing peak GPU memory by around 65% and substantially improving training and sampling throughput. These results suggest the proposed architecture is a practical step toward scalable graph foundation models.

Code

1. Introduction

Graph generation (Simonovsky & Komodakis, 2018; Martinkus et al., 2022; Jo et al., 2024) has advanced substantially with discrete diffusion and flow-based models. A particularly influential line of work, initiated by the diffusion model DiGress (Vignac et al., 2023), and then flow-matching models such as CatFlow (Eijkelboom et al., 2024), DeFoG (Qin et al., 2025a), Cometh (Siraudin et al., 2024), BWFlow (Jiang et al., 2026), and VBFN (Xiong et al., 2026), relies on a specific transformer backbone to denoise categorical node and edge features. This architecture is inspired by the original Graph Transformer (Dwivedi & Bresson, 2021), with the difference that it jointly maintains node, edge, and graph-level features. It uses edge and global information to modulate node attention, updates edge states from attention scores, and pools node-edge information to update global representations. This layered interaction is central to its success in graph generation, where the model must capture both local connectivity and global graph structure.

However, this expressivity comes with a computational cost. Since the model must maintain dense pairwise edge states, both attention computation and dynamic edge representations scale quadratically with the number of nodes, making training expensive under limited GPU resources. A large body of work addresses this bottleneck through sparse attention patterns (Shirzad et al., 2023); hybrid local-global designs (Rampášek et al., 2022); linear and operator-based alternatives (Deng et al., 2024; Sancak et al., 2025); or tokenization-based formulations that cast graphs as sequences of node and edge tokens and apply standard Transformers with minimal graph-specific modifications (Kim et al., 2022). These methods are effective, but typically modify or replace dense self-attention altogether. In graph generation specifically, SparseDiff (Qin et al., 2025b) exploits sparsity to enable training on larger graphs, although this can introduce an unfavorable trade-off with training time.

In this work, we take a complementary route and ask whether the original dense edge-aware architecture can be implemented more efficiently while fully preserving its inductive bias. We focus on the computationally-expensive and memory-hungry attention layer in the original implementation, and design a kernel-friendly version which still modulates attention scores using edge and global features, respects graph masks, and enables edge features update from its attention map. This version can be implemented entirely using `FlexAttention` (Dong et al., 2024), which supports custom attention score modifications and masking. This enables us to use fused attention kernels, bringing throughput and memory gains.

Empirically, our proposed model (FlexAttentionGT) matches the generation quality of DeFoG(Qin et al.,

[*]Equal contribution [1]LTS4, EPFL, Lausanne, Switzerland [2]Università della Svizzera italiana, Lugano, Switzerland. Correspondence to: <first_name.last_name@epfl.ch>.

The Workshop on Graph Foundation Models at the 43rd International Conference on Machine Learning, Seoul, South Korea.
Copyright 2026 by the author(s).

2025a)'s original backbone while reducing peak memory by 65% and improving training and inference throughput by 45% and 94% on average. Ablations show that the edge update is critical: simplifying it causes generation collapse, whereas reducing model size leads to slower convergence and worse final performance. Together, these results demonstrate that edge-aware graph Transformers can be made more efficient while preserving their expressivity, supporting their scalability to the larger graphs encountered in foundation model settings.

2. Background

2.1. Generative Graph Transformer

Recent works in grah generation such as DiGress (Vignac et al., 2023) or BWFlow (Jiang et al., 2026) employ a Graph Transformer (GT) backbone that jointly updates node features $\boldsymbol{X} \in \mathbb{R}^{N \times d_x}$, edge features $\boldsymbol{E} \in \mathbb{R}^{N \times N \times d_e}$, and graph-level features $\boldsymbol{y} \in \mathbb{R}^{d_y}$, with N the number of nodes of the largest graph in a batch. In contrast to a vanilla Transformer (Dwivedi & Bresson, 2021), this backbone injects edge and global features into node attention through FiLM-style score modulation (Perez et al., 2018) (Further details in Appendix C.1). The attention map is then reused to update edge features, while graph-level features are updated from pooled node and edge representations. This design is effective for graph generation because it tightly couples node-, edge-, and graph-level information. However, it bottlenecks efficiency: the model maintains dense pairwise edge states $\boldsymbol{E}$, which need to be included in the node attention, and themselves updated using these attention scores, resulting in quadratic computational and memory costs per layer.

2.2. `FlexAttention`

`FlexAttention` is a PyTorch interface for implementing customized attention variants while lowering them to fused attention kernels (Dong et al., 2024). It exposes two main hooks. The `score_mod` function modifies each pre-softmax attention logit, making it suitable for structural biases and edge-dependent score modulation. The `mask_mod` function specifies which query-key pairs are valid and can be compiled into a block-sparse `BlockMask`, allowing the kernel to skip masked blocks. This interface is attractive for graph Transformers because graph masks and structural score biases can be naturally expressed inside the attention kernel; however, it does not directly support the edge update used in generative graph Transformers. `FlexAttention` returns the attended node values, whereas the original architecture also reuses attention maps to update edge representations. Therefore, adapting this backbone requires separating the kernel-compatible node attention from the attention-dependent edge update.

3. Efficient Attention Layers for Graph Generation

We identify three issues to address when creating a FlexAttention-compatible attention layer: (i) the original architecture uses a non-standard element-wise outer-product attention that is incompatible with efficient attention kernels since they do not give access to the outer product between query and key (ii) the original implementation uses a FiLM layer on these outer products to include edge and global information in the attention mechanism, our layer still needs to include this information in a way that is both compatible with attention kernels but expressive enough so as to not lose performance; and (iii) edge representations depend on the attention scores or the softmax probabilities α, which attention kernels such as FlexAttention do not store in global memory. In the following, we address each of them, with further details in Appendices C.2 and C.3.

From element-wise to scalar dot-product attention. The original backbone computes an unnormalized attention tensor $\boldsymbol{Y} \in \mathbb{R}^{N \times N \times H \times d_f}$ via the element-wise outer product $\boldsymbol{Y}_{ij,h} = \boldsymbol{q}_{i,h} \odot \boldsymbol{k}_{j,h}$, retaining a d_f-dimensional feature vector per edge and head. Softmax is then applied *independently per channel*, yielding d_f separate attention distributions per query-head pair. While expressive, this formulation cannot be lowered to a standard fused kernel. We replace it with standard scaled dot-product attention. The unnormalized scalar logit for each head h is

$$s_{ij,h} = \frac{\boldsymbol{q}_{i,h} \cdot \boldsymbol{k}_{j,h}}{\sqrt{d_f}} \in \mathbb{R}, \tag{1}$$

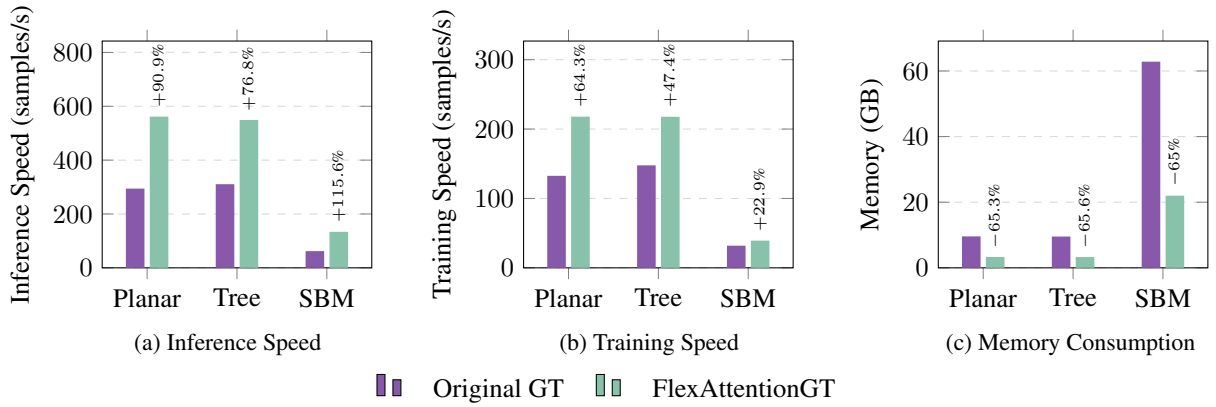
giving a single attention distribution per query-head pair and making the node-update compatible with standard FlexAttention.

Head-wise structural FiLM layer To inject edge features $\boldsymbol{E} \in \mathbb{R}^{N \times N \times d_e}$ and global features $\boldsymbol{y} \in \mathbb{R}^{d_y}$ into attention, we project them to head-wise scalars:

$$s'_{ij,h} = s_{ij,h} \cdot \left(E^{\text{mul}}_{ij,h} + 1\right) + E^{\text{add}}_{ij,h}, \tag{2}$$

where $\boldsymbol{E}^{\text{mul}}, \boldsymbol{E}^{\text{add}} \in \mathbb{R}^{N \times N \times H}$ are obtained by linearly projecting E and y to H-dimensional scalars. This head-wise FiLM-style modulation is applied *inside* the fused kernel via the `score_mod` hook, so no quadratic tensor is written to global memory during the attention pass. Padding and graph masks are handled via a `BlockMask`, allowing the kernel to skip fully-masked blocks.

Single-pass node and edge update via $\boldsymbol{I_N}$ concatenation. A key challenge is that edge representations in the original backbone are updated from pre-softmax attention scores, which FlexAttention does not return. Instead, we use the



(a) Inference Speed (b) Training Speed (c) Memory Consumption

Figure 1. Computational efficiency across datasets. Percentages denote the relative change compared to the original implementation.

normalized attention coefficients $\boldsymbol{\alpha} \in \mathbb{R}^{N\times N\times H}$ and recover them in a *single* FlexAttention call by concatenating an identity matrix $\boldsymbol{I_N} \in \mathbb{R}^{N\times N}$ to the value matrix:

$$\mathrm{FlexAttn}(\boldsymbol{Q}, \boldsymbol{K}, [\boldsymbol{V} \,\|\, \boldsymbol{I_N}]) = \boldsymbol{\alpha}[\boldsymbol{V} \,\|\, \boldsymbol{I_N}] = [\boldsymbol{\alpha V} \,\|\, \boldsymbol{\alpha}]. \tag{3}$$

The kernel simultaneously produces the aggregated node values $\boldsymbol{V'} = \boldsymbol{\alpha V} \in \mathbb{R}^{N\times H\times d_f}$ and the attention maps $\boldsymbol{\alpha} \in \mathbb{R}^{N\times H\times N}$ as the trailing output channels. Edge representations are updated using:

$$E'_{ij} = \mathrm{Linear}(\alpha_{ij} \odot \phi(\boldsymbol{y})) + E_{ij}, \tag{4}$$

where $\phi(\boldsymbol{y})$ provides a FiLM scale and bias learned from the global features $\boldsymbol{y}$. Node and edge updates thus share a single function call. This works well for the benchmark datasets we consider here which contain graphs below 256 nodes. For larger graphs, the concatenation of n channels to the value vectors increases the SRAM requirements, which requires us to decrease the block size and leads to worse throughput. In these cases, we calculate the attention output and the attention map in two separate passes, which allows us to keep an appropriate block size. We address this in more details in Appendix B.2.

The edge update remains quadratic in N, but the dominant constant is reduced because the heavy attention computation itself is handled by FlexAttention.

4. Experiments

We train our FlexAttention-based architecture and the original GT backbone using the DeFog (Qin et al., 2025a) framework, and evaluate on the Planar, SBM, and Tree datasets originally used in DeFoG, while keeping all training hyperparameters fixed across configurations. Performance is assessed using validity, uniqueness, and novelty (V.U.N.↑), as well as the Ratio↓ metric comparing graph-statistic distances between generated and test graphs relative to the train–test baseline.

4.1. Graph generation performance

Table 1 reports generation quality across all datasets. Our updated architecture matches or slightly improves upon the original GT on V.U.N. metrics, including a small gain on SBM. Although the Ratio metric is marginally higher on Planar and Tree, the differences remain within one standard deviation, indicating comparable structural fidelity overall.

Table 1. Generation performance on Planar, Tree, and SBM.

Model	V.U.N.↑	Ratio↓
	Planar	
Train set	100	1.00
GT (Original)	99.5 ± 1.0	2.47 ± 0.33
FlexAttentionGT (Ours)	$\mathbf{100.0 \pm 0.0}$	$\mathbf{2.21 \pm 0.95}$
	Tree	
Train set	100	1.00
GT (Original)	$\mathbf{97.0 \pm 3.7}$	2.24 ± 0.63
FlexAttentionGT (Ours)	$\mathbf{97.0 \pm 1.9}$	$\mathbf{2.12 \pm 0.87}$
	SBM	
Train set	85.9	1.00
GT (Original)	85.0 ± 5.3	$\mathbf{3.39 \pm 1.08}$
FlexAttentionGT (Ours)	$\mathbf{89.5 \pm 5.7}$	3.70 ± 1.08

4.2. Speedup and memory usage improvement

Figure 1 compares inference throughput, training speed, and peak GPU memory across datasets. Our model is substantially faster than the original on both training and inference, with speedups ranging from 23% to 116% depending on the dataset and regime. Peak memory is reduced by approximately 65% across all datasets, roughly one third of the original. These gains are consistent across datasets, indicating that the efficiency improvements are not dataset-specific.

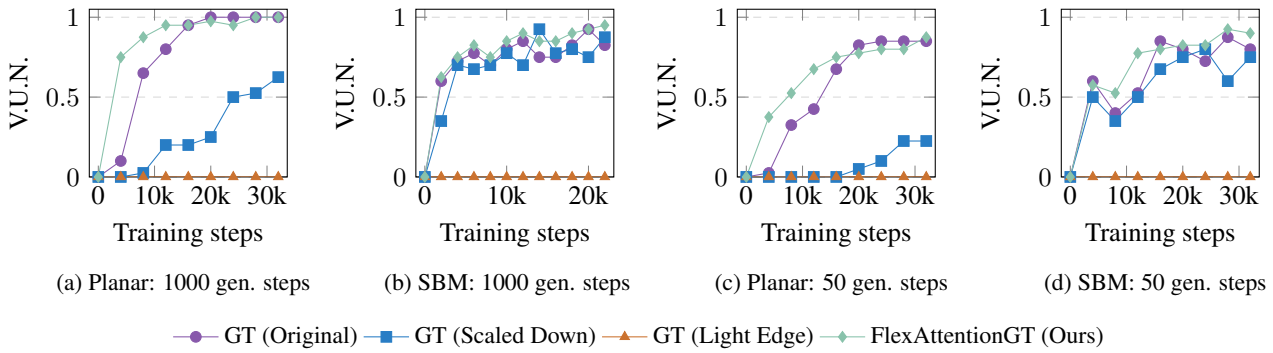


Figure 2. V.U.N. curves over training steps on the Planar and SBM datasets, comparing FlexAttentionGT against the original GT and its lightweight variants. We evaluate 1000 and 50 generation steps, respectively.

4.3. Ablations

To isolate the contribution of our key design choices, we ablate three dimensions: the number of generation steps (1000 vs. 50), model size via a scaled-down version of the original implementation (**Scaled Down**), and the edge update strategy via a variant without per-edge modulation of attention scores (**Light Edge**), both matched to our architecture in peak memory. All other hyperparameters are kept identical.

Generation quality across training with 1000 steps. Figures 2a and 2b show V.U.N. curves on Planar and SBM with 1000 generation steps. Our model matches or improves on all baselines on convergence and final performance. GT (Scaled Down) learns more slowly and underperforms, showing that matching memory by reducing capacity is a poor trade-off. GT (Light Edge) fails to learn entirely, confirming that per-edge modulation of attention scores is a critical component. The effect is most pronounced on Planar, but our model also converges fastest on SBM.

Generation quality across training with 50 steps. Figures 2c and 2d repeat the same comparison under a reduced budget of 50 generation steps. Under this constrained setting, GT (Light Edge) again fails to learn, and GT (Scaled Down) also fails in the Planar dataset. The original GT shows moderate but inconsistent performance, while our model achieves the best and most stable results across training, demonstrating improved sample efficiency and robustness in the low-step regime.

Training dynamics. Figure 3 shows training loss curves on Planar for the original and updated models. The two curves overlap closely throughout training, indicating that the architectural changes do not alter the optimization landscape. This suggests that the observed quality differences are unlikely to originate from a change in how the model optimizes, but rather a benefit of the inductive biases of our attention layer.

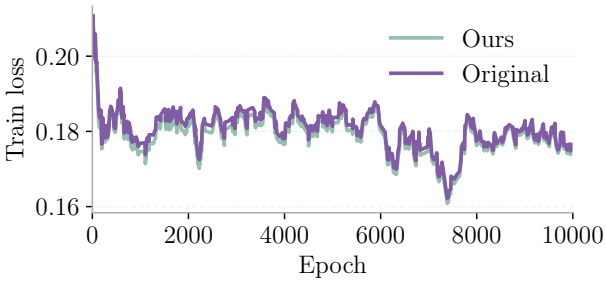


Figure 3. Training loss comparison between the updated architecture and the original one, smoothed with a window size of 20.

Edge update using normalized attention coefficients. Our edge updates uses the normalized attention coefficients $\alpha_{ij,h}$ recovered using (Eq. 3), whereas the baseline implementation uses unnormalized attention scores. To isloate the effect of this change, we implement a version of FlexAttentionGT which separately calculates the unnormalized attention coefficients and uses those for edge updates. We train this version on the planar dataset and report the V.U.N curves with 1000 generation steps for both implementations in Figure 4. The unnormalized variant converges more slowly, showing that the normalization provided by softmax is beneficial for stable edge feature learning.

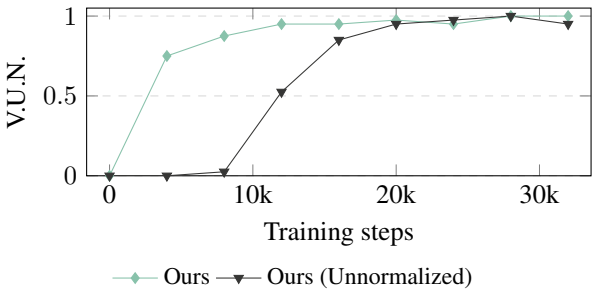


Figure 4. V.U.N. curves comparing normalized and unnormalized networks over training steps on the Planar dataset, evaluated with 1000 generation steps.

5. Conclusion

In this work, we presented an efficient and expressive edge-aware graph Transformer that leverages recent fused attention kernels for graph generation. By shifting from element- to head-wise FiLM transformations, we eliminated major memory bottlenecks while preserving nuanced structural bias injection. Furthermore, our approach allows us to seamlessly extract the attention map and concurrently update edge features within a single kernel pass.

Our empirical benchmarks across Planar, Tree, and SBM datasets demonstrate that this architectural refinement yields an average of $\mathbf{44.8}\%$ speedup in training, an average of $\mathbf{94.4}\%$ speedup in inference, and consumes $\mathbf{65}\%$ less peak GPU memory compared to the original dense implementation, with no loss in generation fidelity. These gains can be particularly relevant in the context of graph foundation models, where scaling edge-aware architectures to larger and more diverse graphs is a key challenge.

Beyond graph generation, the design principles introduced here (head-wise structural conditioning inside a fused kernel and single-pass node-edge updates) could be applicable to any edge-aware transformer setting, including those used in graph classification, regression, and link prediction, offering a practical path toward more scalable graph learning.

Acknowledgements

This project was supported by the Swiss National Science Foundation (SNSF grant 10001445). Vincent Jung is partially supported by the Swiss National Science Foundation (grant 205121 207437: C-LING) and the Fondazione Aldo e Cele Daccò.

Use of Large Language Models

Large Language Models (LLMs) were used as a coding and writing assistant tool in the preparation of this manuscript and its experiments. Specifically, they were employed to aid in writing, checking spelling, and editing for clarity. They were also used to assist in coding and plotting of the results. Nonetheless, all text and code produced with the assistance of LLMs was carefully reviewed, verified, and revised by the authors to ensure accuracy and appropriateness. The use of LLMs was limited to these support functions, and the authors take full responsibility for the final contents.

Impact Statement

The primary objective of this paper is to advance graph generation with a more efficient network. While this development has the potential to bring about both positive and negative societal or ethical impacts, particularly in areas like biomedical and chemical research, we currently do not foresee any immediate societal concerns associated with the proposed methodology.

References

Buterez, D., Janet, J. P., Oglic, D., and Lio, P. An end-to-end attention-based approach for learning on graphs. *arXiv preprint arXiv:2402.10793*, 2024. URL https://arxiv.org/abs/2402.10793.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *arXiv preprint arXiv:2205.14135*, 2022. URL https://arxiv.org/abs/2205.14135.

Deng, C., Yue, Z., and Zhang, Z. Polynormer: Polynomial-expressive graph transformer in linear time. *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2403.01232.

Diao, C. and Loynd, R. Relational attention: Generalizing transformers for graph-structured tasks. *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2210.05062.

Dong, J., Feng, B., Guessous, D., Liang, Y., and He, H. FlexAttention: A programming model for generating optimized attention kernels. *arXiv preprint arXiv:2412.05496*, 2024. URL https://arxiv.org/abs/2412.05496.

Dwivedi, V. P. and Bresson, X. A generalization of transformer networks to graphs. *AAAI Conference on Artificial Intelligence (AAAI)*, 2021. URL https://arxiv.org/abs/2012.09699.

Eijkelboom, F., Bartosh, G., Naesseth, C. A., Welling, M., and van de Meent, J.-W. Variational Flow Matching for Graph Generation. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2406.04843.

Hussain, M. S., Zaki, M. J., and Bhatt, D. Global self-attention as a replacement for graph convolution. *Proceedings of the 28th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2022. URL http://dx.doi.org/10.1145/3534678.3539296.

Jiang, K., Cui, J., Dong, X., and Toni, L. Bures-wasserstein flow matching for graph generation. *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2506.14020.

Jo, J., Kim, D., and Hwang, S. J. Graph Generation with Diffusion Mixture. *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2302.03596.

Kim, J., Nguyen, T. D., Min, S., Cho, S., Lee, M., Lee, H., and Hong, S. Pure transformers are powerful graph learners. *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2207.02505.

Kong, L. et al. Generalized-distance transformer: A fine-grained analysis of graph transformer design. *arXiv preprint arXiv:2511.08028*, 2025. URL https://arxiv.org/abs/2511.08028.

Li, C., Chen, J., Hopcroft, J. E., and He, K. DAM-GT: Dual positional encoding-based attention masking graph transformer for node classification. *arXiv preprint arXiv:2505.17660*, 2025. URL https://arxiv.org/abs/2505.17660.

Ma, L., Pal, S., Zhang, Y., Torr, P. H., and Coates, M. Plain transformers can be powerful graph learners. *arXiv preprint arXiv:2504.12588*, 2025. URL https://arxiv.org/abs/2504.12588.

Martinkus, K., Loukas, A., Perraudin, N., and Wattenhofer, R. SPECTRE: Spectral Conditioning Helps to Overcome the Expressivity Limits of One-shot Graph Generators. *International Conference on Machine Learning (ICML)*, 2022. URL https://arxiv.org/abs/2204.01613.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. FiLM: Visual reasoning with a general conditioning layer. *AAAI Conference on Artificial Intelligence (AAAI)*, 2018. URL https://arxiv.org/abs/1709.07871.

Qin, Y., Madeira, M., Thanou, D., and Frossard, P. DeFoG: Discrete Flow Matching for Graph Generation. *International Conference on Machine Learning (ICML)*, 2025a. URL https://arxiv.org/abs/2410.04263.

Qin, Y., Vignac, C., and Frossard, P. SparseDiff: Sparse Discrete Diffusion for Scalable Graph Generation. *Transactions on Machine Learning Research (TMLR)*, 2025b. URL https://arxiv.org/abs/2311.02142.

Qiu, S. et al. FlexGNN: A high-performance, large-scale full-graph GNN system with best-effort training plan optimization. *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2025. doi: 10.1145/3711896.3736964. URL https://dl.acm.org/doi/10.1145/3711896.3736964.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2205.12454.

Sancak, K., Hua, Z., Fang, J., Xie, Y., Malevich, A., Long, B., Balin, M. F., and Ümit V. Çatalyürek. A scalable and effective alternative to graph transformers. *AAAI Conference on Artificial Intelligence (AAAI)*, 2025. URL https://ojs.aaai.org/index.php/AAAI/article/view/34231/36386.

Shehzad, A., Xia, F., Abid, S., Peng, C., Yu, S., Zhang, D., and Verspoor, K. Graph transformers: A survey. *arXiv preprint arXiv:2407.09777*, 2024. URL https://arxiv.org/abs/2407.09777.

Shirzad, H., Velingker, A., Venkatachalam, B., Sutherland, D. J., and Smola, A. J. Exphormer: Sparse transformers for graphs. *International Conference on Machine Learning (ICML)*, 2023. URL https://arxiv.org/abs/2303.06147.

Simonovsky, M. and Komodakis, N. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. *Artificial Neural Networks and Machine Learning (ICANN)*, 2018. URL https://arxiv.org/abs/1802.03480.

Siraudin, A., Malliaros, F. D., and Morris, C. Cometh: A continuous-time discrete-state graph diffusion model. *arXiv preprint arXiv:2406.06449*, 2024. URL https://arxiv.org/abs/2406.06449.

Tomczak, N. and Kuppannagari, S. Longer attention span: Increasing transformer context length with sparse graph processing techniques. *arXiv preprint arXiv:2502.01659*, 2025. URL https://arxiv.org/abs/2502.01659.

Vignac, C., Krawczuk, I., Siraudin, A., Wang, B., Cevher, V., and Frossard, P. Digress: Discrete denoising diffusion for graph generation. *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2209.14734.

Xiong, Y., Chen, J., Gong, X., Wu, J., Pan, S., and Hu, W. Variational bayesian flow network for graph generation. *arXiv preprint arXiv:2601.22524*, 2026. URL https://arxiv.org/abs/2601.22524.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform bad for graph representation? In *Advances in Neural Information Processing Systems*, 2021.

Yuan, C., Zhao, K., Kuruoglu, E. E., Wang, L., Xu, T., Huang, W., Zhao, D., Cheng, H., and Rong, Y. A survey of graph transformers: Architectures, theories and applications. *arXiv preprint arXiv:2502.16533*, 2025. URL https://arxiv.org/abs/2502.16533.

Zhang, Z., Wang, X., Guan, C., Zhang, Z., Li, H., and Zhu, W. Autogt: Automated graph transformer architecture search. In *The Eleventh International Conference on Learning Representations*, 2023.

Zhang, Z. et al. A closer look at graph transformers: Cross-aggregation and beyond. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://openreview.net/forum?id=7FhWZFoVem.

Zhuo, J., Liu, Y., Lu, Y., Ma, Z., Fu, K., Wang, C., Guo, Y., Wang, Z., Cao, X., and Yang, L. Dualformer: Dual graph transformer. In *International Conference on Learning Representations*, 2025.

A. Related Work

In the following, we take a deeper look and situate our work within the broader literature on Graph Transformers. We cover structural attention biases, attention masking, efficient attention mechanisms, and more recent graph transformer architectures motivated by graph foundation models. We refer the reader to recent surveys for a more exhaustive and complementary overview (Shehzad et al., 2024; Yuan et al., 2025).

A.1. Graph Transformers with Structural Attention Biases

A central challenge in applying Transformers to graphs is injecting graph-structural information into the attention computation. Graphormer (Ying et al., 2021) introduced centrality, spatial, and edge encodings that modulate pre-softmax logits, enabling standard dot-product attention to be directly conditioned on shortest-path distances and edge features, showing that well-designed structural biases suffice to bring a vanilla Transformer to state-of-the-art performance on graph-level tasks. Edge-augmented Graph Transformer (EGT; Hussain et al. 2022) proposed explicit residual *edge channels* that are bidirectionally coupled with node attention: edge embeddings bias the attention weights, and the resulting pre-softmax scores in turn update the edge representations. This bidirectional coupling is architecturally close to the generative GT backbone built on (Dwivedi & Bresson, 2021), differing mainly in its use of FiLM-style modulation and the joint maintenance of graph-level features. Relational Attention (Diao & Loynd, 2023) generalizes Transformer attention to jointly consider and update edge vectors at each layer, achieving strong results on algorithmic reasoning benchmarks. Finally, the Generalized-Distance Transformer (GDT; Kong et al. 2025) offers a unified analysis of structural encodings and attention mechanisms across a large evaluation suite, distilling design choices that consistently transfer across tasks and scales.

Our work shares the same spirit of preserving dense edge representations, but focuses instead on making the resulting architecture compatible with hardware-fused attention kernels without sacrificing expressivity.

A.2. Graph Transformers with Adjacency Masking

An alternative to score-level structural biases is to *mask* the attention matrix so that each node attends only to a structured neighborhood. Exphormer (Shirzad et al., 2023) combines sparse virtual global nodes with local neighborhood attention, compiling the resulting block-sparse patterns into efficient kernels; its BlockMask abstraction directly inspired our use of `mask_mod` for handling padding and graph boundaries. GraphGPS (Rampášek et al., 2022) introduced a hybrid architecture that runs a local MPNN and a global Transformer in parallel, concatenating their outputs at each layer; this recipe has become a strong baseline for graph learning precisely because it avoids committing entirely to either sparse or dense attention. Masked Attention for Graphs (Buterez et al. 2024) explores graphs as edge sets and vertically interleaves masked and vanilla self-attention modules, demonstrating that simple connectivity-based masking can match or exceed more complex positional encoding schemes on long-range benchmarks. More recently, DAM-GT (Li et al., 2025) introduced a dual positional encoding scheme with an attribute-aware masking strategy specifically designed to prevent attention diversion in neighborhood-token architectures for node classification.

A.3. Scalable and Efficient Attention for Graphs

Reducing the quadratic computational and memory cost associated with dense attention has become a major focus of research in recent years, tackled from several complementary angles.

Linear and operator-based alternatives. Polynormer (Deng et al., 2024) achieves linear-time attention through a polynomial-expressive formulation, while GECO (Sancak et al., 2025) proposes a scalable message-passing alternative to full graph Transformers. DUALFormer (Zhuo et al., 2025) introduces a dual-dimensional design that performs self-attention in the feature dimension rather than the node dimension, achieving scalability through linear approximation. UGCFormer (Zhang et al., 2025) identifies a shared *cross-aggregation* mechanism underlying existing efficient Graph Transformers and proposes a linearized dual cross-attention module (DCA) to realize it. Powerful Plain Graph Transformer (PPGT, Ma et al. 2025) demonstrates that three minimal modifications to the standard Transformer (L_2 attention, adaptive RMS normalization, and a relative positional bias) suffice to achieve competitive graph learning without any message-passing components, narrowing the gap between general-purpose and graph-specific architectures.

Sparse implementations. Tomczak & Kuppannagari (Tomczak & Kuppannagari, 2025) treat sparse attention masks as graph adjacency patterns and use classical graph processing algorithms to obtain work-optimal attention implementations;

their framework complements our fused-kernel approach by targeting scenarios where true sparsity can be exploited. In the graph-generation setting specifically, SparseDiff (Qin et al., 2025b) exploits graph sparsity to scale discrete diffusion to larger graphs, but at the cost of increased training time. FlexGNN (Qiu et al., 2025) addresses full-graph GNN training at scale by managing intermediate data and optimizing inter-GPU communication plans, and is orthogonal to our work on attention kernel efficiency.

Towards graph foundation models. GraphBFF (**?**) is the first end-to-end recipe for billion-parameter Graph Foundation Models on heterogeneous graphs. In particular, they present the GraphBFF transformer, which replaces dense attention with type-aware sparse attention and degree-constrained neighbor sampling, enabling scalable learning on highly heterogeneous, large-scale graphs. The need for these efficiency mechanisms highlights the importance of reducing the memory cost of dense, edge-aware graph architectures.

A.4. Architecture Search and Tokenization-Based Methods

AutoGT (Zhang et al., 2023) proposes an automated search over the joint space of graph encoding strategies and Transformer architectures, unifying most existing designs into a single searchable framework. Kim et al. (2022) reformulate graphs as sequences of node and edge tokens and apply a standard Transformer with minimal graph-specific modifications, highlighting that carefully designed tokenization can compensate for the absence of explicit structural biases. The GDT analysis (Kong et al., 2025) additionally validates these tokenization-based formulations through extensive few-shot transfer experiments.

B. Efficiency Benchmarks

B.1. Detailed efficiency benchmarks on Planar, Tree and SBM

In this section, we provide the exact numerical values corresponding to the barplots presented in Section 4. Table 2 summarizes the inference speed, training speed, and peak GPU memory consumption during training for all three implementations across the evaluated datasets.

Table 2. Detailed computational efficiency results for the Planar, Tree, and SBM datasets. Inference and training speeds are measured in samples per second. Memory is reported during training, in MB. Tested on A100 GPU.

	Inference (samples/s) ↑		Training (samples/s) ↑		Memory (MB) ↓	
Dataset	Orig.	Upd.	Orig.	Upd.	Orig.	Upd.
Planar	294.2	561.5	132.6	217.9	9,580	3,323
Tree	310.6	549.1	147.7	217.7	9,538	3,281
SBM	62	133.7	31.9	39.2	62,840	21,991

B.2. Efficiency benchmark on larger graphs

The I_N concatenation trick used in our single-pass edge update strategy increases the SRAM requirements for each block in the FlexAttention kernel. For large graphs above 256 nodes, this forces us to reduce the block size leading to a slowdown because of the larger number of kernel calls. We implement a two-pass version of our architecture where the attention output and the attention map are calculated in two separate passes, allowing us to keep a larger block size. We compare the throughput and memory requirement of this implementation with the single-pass version on the SBM dataset generated with graphs of size 400 to 600 nodes. The results are shown in Table 3. We observe that the two-pass version achieves higher throughput than the single-pass version while being mathematically equivalent, and still delivering large gains compared to the original implementation.

Future work can explore custom kernels which perform the update of the attention map using input edge features and the subsequent edge feature updates using said attention map in a block-wise manner, removing the need to extract the entire attention map to global memory and thus preserving the linear memory scaling of FlexAttention even with the edge update. This should be feasible since the inclusion of edge features and the edge update are both pointwise operations that can be performed for a pair of nodes (i,j) independently of other pairs, at least when using unnormalized attention coefficients for the edge updates. This could not only reduce memory requirements due to the attention calculation itself, but also reduce the amount of intermediate edge features that need to be stored in global memory, which become the major bottleneck for large graphs.

Table 3. Efficiency benchmark on the SBM-Large dataset (batch size 2), with an average size of 500 nodes. Inference and training speeds are measured in samples per second. Memory is reported in MB.Peak GPU memory in MB. Tested on A100 GPU.

	Inference		Training	
Model	Samples/s ↑	Mem (MB) ↓	Samples/s ↑	Mem (MB) ↓
Orig.	6.6	10,412	3.4	43,429
Updated (single-pass)	13.2	2,584	0.9	15,320
Updated (multi-pass)	13.7	2,595	6.1	15,316

C. Graph Transformer Layer Details

We describe the two layer variants in full detail. Throughout, let N denote the number of nodes, H the number of attention heads, and $d_f = d_x/H$ the per-head feature dimension. We write $X \in \mathbb{R}^{N \times d_x}$, $E \in \mathbb{R}^{N \times N \times d_e}$, and $y \in \mathbb{R}^{d_y}$ for the node, edge, and global feature matrices respectively. All linear projections are written as $\mathrm{Linear}(\cdot)$ with output dimensions indicated in context. We use $\odot$ for element-wise multiplication and $[\,\cdot \,\|\, \cdot\,]$ for concatenation along the last axis.

C.1. Original Layer

Query, key, and value projections. Node features are projected to queries, keys, and values:

$$Q = \mathrm{Linear}_{d_x}(X), \quad K = \mathrm{Linear}_{d_x}(X), \quad V = \mathrm{Linear}_{d_x}(X), \tag{5}$$

each of shape $\mathbb{R}^{N \times d_x}$, then reshaped to $\mathbb{R}^{N \times H \times d_f}$.

Element-wise outer-product attention. Rather than computing a scalar dot product, the original layer forms a d_f-dimensional bilinear tensor per edge:

$$Y_{ij,h} = \frac{q_{i,h} \odot k_{j,h}}{\sqrt{d_f}} \in \mathbb{R}^{d_f}, \tag{6}$$

yielding $Y \in \mathbb{R}^{N \times N \times H \times d_f}$. This retains the full per-channel interaction between $q_{i,h}$ and $k_{j,h}$.

Edge FiLM on attention scores. Edge features are projected to per-channel multiplicative and additive modifiers:

$$E^{\mathrm{mul}} = \mathrm{Linear}_{d_x}(E), \quad E^{\mathrm{add}} = \mathrm{Linear}_{d_x}(E), \tag{7}$$

both reshaped to $\mathbb{R}^{N \times N \times H \times d_f}$, and applied as a FiLM transform (Perez et al., 2018) over the full feature dimension:

$$Y_{ij,h} \leftarrow Y_{ij,h} \odot \left(E^{\mathrm{mul}}_{ij,h} + 1\right) + E^{\mathrm{add}}_{ij,h}. \tag{8}$$

Edge update. The modulated tensor Y is flattened to $\mathbb{R}^{N \times N \times d_x}$ and conditioned on global features via a second FiLM transform:

$$\hat{E}_{ij} = \mathrm{Linear}_{d_y}(y) + \left(\mathrm{Linear}_{d_y}(y) + 1\right) \odot Y^{\mathrm{flat}}_{ij}, \tag{9}$$

followed by a final linear projection to $\mathbb{R}^{d_e}$.

Per-channel softmax and node update. Softmax is applied over the key dimension independently for each channel:

$$\alpha_{ij,h,c} = \frac{\exp(Y_{ij,h,c})}{\sum_{j'} \exp(Y_{ij',h,c})}, \quad c = 1, \ldots, d_f, \tag{10}$$

yielding d_f independent attention distributions per query-head pair. The weighted sum over values is then:

$$\hat{x}_{i,h,c} = \sum_j \alpha_{ij,h,c} \cdot v_{j,h,c}, \tag{11}$$

and the result is flattened to $\mathbb{R}^{N \times d_x}$.

Global FiLM on node update and node output. Global features modulate the aggregated node representations before the output projection:

$$\hat{X} \leftarrow \mathrm{Linear}_{d_y}(y) + \big(\mathrm{Linear}_{d_y}(y) + 1\big) \odot \hat{X}. \tag{12}$$

Global feature update. The global representation is updated by pooling node and edge features:

$$y \leftarrow \mathrm{MLP}\big(y + \mathrm{Pool}(X) + \mathrm{Pool}(E)\big), \tag{13}$$

where Pool denotes principal neighborhood aggregation (PNA) over nodes and edges respectively.

C.2. FlexAttention Layer

The FlexAttention layer preserves the three-stream structure of the original but replaces the element-wise outer product with standard dot-product attention, compresses the FiLM modulation to head-wise scalars, and recovers the attention map for the edge update in a single function call. The overall architecture is illustrated in Figure 5.

Query, key, and value projections. Projections are identical to the original:

$$Q, K, V = \mathrm{Linear}_{d_x}(X), \quad \text{each} \in \mathbb{R}^{N \times H \times d_f}, \tag{14}$$

then transposed to $\mathbb{R}^{H \times N \times d_f}$ as required by the FlexAttention interface.

Scalar dot-product attention logits. The unnormalized attention logit for head h is a scalar:

$$s_{ij,h} = \frac{q_{i,h} \cdot k_{j,h}}{\sqrt{d_f}} \in \mathbb{R}. \tag{15}$$

Head-wise structural FiLM via `score_mod`. Edge and global features are projected to head-wise scalar modifiers:

$$E_{\mathrm{mul}} = \mathrm{Linear}_H(E) \in \mathbb{R}^{N \times N \times H}, \tag{16}$$

$$E_{\mathrm{bias}} = \mathrm{Linear}_H(E) \in \mathbb{R}^{N \times N \times H}, \tag{17}$$

$$y_{\mathrm{mul}} = \mathrm{Linear}_H(y) \in \mathbb{R}^{H}, \tag{18}$$

$$y_{\mathrm{bias}} = \mathrm{Linear}_H(y) \in \mathbb{R}^{H}. \tag{19}$$

These are combined into a multiplicative modifier $M \in \mathbb{R}^{N \times N \times H}$ and an additive bias $B \in \mathbb{R}^{N \times N \times H}$:

$$M_{ij,h} = \sigma\big(E_{\mathrm{mul},ij,h} + y_{\mathrm{mul},h}\big), \quad B_{ij,h} = \sigma\big(E_{\mathrm{bias},ij,h} + y_{\mathrm{bias},h}\big), \tag{20}$$

where σ denotes the softsign function $\sigma(z) = z/(1 + |z|)$, used to bound the modifiers and stabilise training. The modified score passed to the fused kernel is:

$$s'_{ij,h} = \texttt{score_mod}(s_{ij,h}) = s_{ij,h} \cdot (1 + M_{ij,h}) + B_{ij,h}. \tag{21}$$

Graph padding masks are handled via a `BlockMask`, which instructs the kernel to skip fully-masked blocks and enforces that $s'_{ij,h} \to -\infty$ for invalid pairs.

Single-pass node and edge update via I_N concatenation. FlexAttention does not materialize the softmax probabilities $\alpha_{ij,h}$ in global memory. To recover them for the edge update without a second kernel pass, we concatenate the $N \times N$ identity matrix $I \in \mathbb{R}^{N \times N}$ to the value matrix along the feature axis:

$$\mathrm{FlexAttn}(Q,\, K,\, [V \,\|\, I]) = \alpha\,[V \,\|\, I] = [\,\underbrace{\alpha V}_{\hat{X}} \,\|\, \underbrace{\alpha}_{\alpha}\,], \tag{22}$$

where $\alpha \in \mathbb{R}^{N \times N \times H}$ are the softmax-normalised attention weights. The first d_f output channels give the aggregated node values $\hat{X} \in \mathbb{R}^{N \times d_x}$, and the trailing N channels recover the full attention map α. Node and edge representations are thus both obtained from a single invocation of the fused kernel.

Edge update. The recovered attention map $\alpha \in \mathbb{R}^{N\times N\times H}$ is projected to the edge feature space and conditioned on global features:

$$E'_{ij} = \mathrm{Linear}_{d_e}\big(\big(\mathrm{Linear}_{d_x}(y)+1\big)\odot W_{\mathrm{head}}\,\alpha_{ij}\big) + \mathrm{Linear}_{d_e}(y) + E_{ij}, \tag{23}$$

where $W_{\mathrm{head}} \in \mathbb{R}^{d_x\times H}$ lifts the H head scalars to d_x before the global FiLM and final projection. The two linear maps and the y-FiLM are fused algebraically into a single `einsum`, avoiding an explicit $N^2 \times d_x$ intermediate tensor.

Global feature update. Identical to the original layer:

$$y \leftarrow \mathrm{MLP}\big(y + \mathrm{PNA}(X) + \mathrm{PNA}(E)\big). \tag{24}$$

Comparison summary. Table 4 summarizes the key differences between the two layer variants.

Table 4. Comparison of the original and FlexAttention layer variants.

	Original	**FlexAttention**
Attention logit	$q_{i,h} \odot k_{j,h} \in \mathbb{R}^{d_f}$	$q_{i,h} \cdot k_{j,h} \in \mathbb{R}$
Softmax	Per-channel (d_f distributions per (i,h))	Standard (1 distribution per (i,h))
Edge FiLM bandwidth	$d_x = H \cdot d_f$ channels	H scalars
Edge FiLM location	Applied to $Y \in \mathbb{R}^{N\times N\times H\times d_f}$	Inside fused kernel via `score_mod`
Edge update source	Full d_x-dim bilinear Y^{flat}	H-dim attention map α
Node & edge update	Two separate passes	Single fused kernel call

C.3. Further FlexAttention details

FlexAttention uses a FlashAttention-style (Dao et al., 2022) algorithm, tiling the query, key and value matrices and using the online softmax trick to never materialize the whole attention matrix at once which enables it to have a linear memory complexity with the number of queries/keys/values. The size of the tiling blocks and the specific backend that FlexAttention uses depends on the GPU type, available kernels and hidden dimensions of the query, key and value vectors.

The tiled blocks of the query, key and value matrix have to fit into SRAM, which is limited and depends on the GPU. We have found good performance with blocks of size 32x32, but these parameters may need to be changed depending on the environment available and dataset used. Moreover, on recent GPUs, FlexAttention will prioritize using FlashAttention 4 as a backend. Due to the identity matrix concatenation to the value matrix, its hidden dimension can be different from a power of 2 which FlashAttention 4 does not deal well with. We found more success forcing the backend to be Triton with the `kernel_options` argument in FlexAttention.

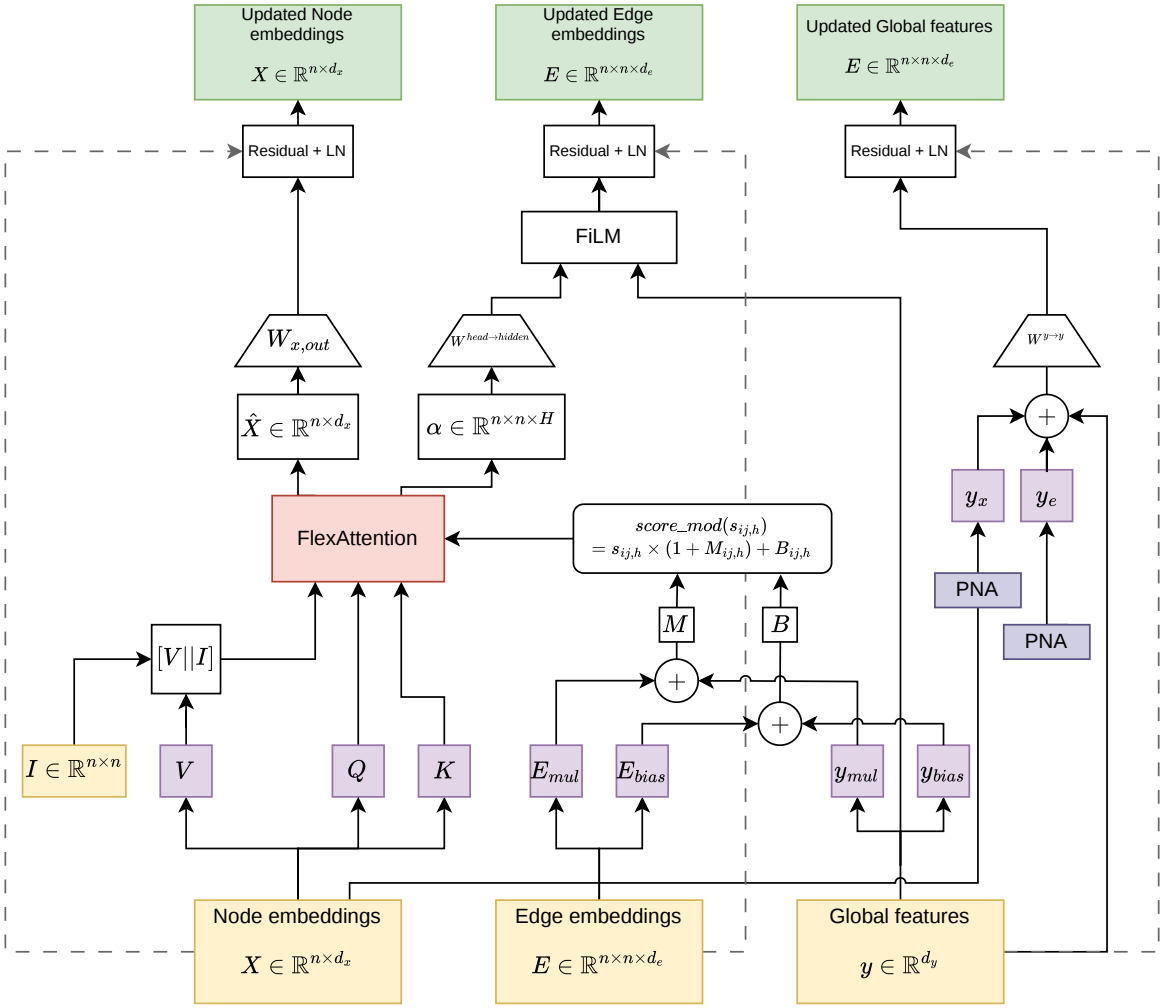


Figure 5. Schematic of the FlexAttentionGT architecture.

D. Experiments Setup.

Dataset size We report the dataset size in Table 5.

Table 5. Sizes for different dataset we use in experiments.

Dataset	Min Nodes	Max Nodes
Planar	64	64
Tree	64	64
SBM	44	187
SBM-Large	401	599

Sampling hyperparameters and training setup. We report in Table 6 the machine used for training, the number of training epochs, and the sampling hyperparameters for the original GT architecture and our FlexAttention version. These settings correspond to the results in Table 1 and the training curves in Figure 2. For both the 50-step and 1000-step sampling curves, we evaluate checkpoints throughout training using the same sampling hyperparameters as in the official DeFoG implementation. All other training settings also follow the default configuration of the official DeFoG implementation. For the final reported results in 1, we select the best-performing hyperparameters for each model.

Table 6. Training and sampling settings for Planar, Tree, and SBM.

Model	Initial Distribution	Train Distortion	Sample Distortion	ω	η	GPU Type	Training Epochs
			Planar				
Original	Marginal	Polydec	Polydec	50	0.05	V100	34000
Ours	Marginal	Polydec	Polydec	50	0.02	V100	34000
			Tree				
Original	Marginal	Polydec	Polydec	0	0	V100	26000
Ours	Marginal	Polydec	Polydec	0	0	V100	16500
			SBM				
Original	Absorbing	Identity	Identity	0.0	0.0	GH200	28000
Ours	Absorbing	Identity	Identity	0.0	0.0	GH200	28000