# Towards Principled Graph Transformers

**Luis Müller**
RWTH Aachen University
luis.mueller@cs.rwth-aachen.de

**Daniel Kusuma**
RWTH Aachen University

**Blai Bonet**
Universitat Pompeu Fabra

**Christopher Morris**
RWTH Aachen University

## Abstract

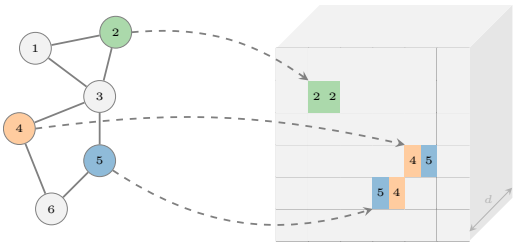
The expressive power of graph learning architectures based on the $k$-dimensional Weisfeiler–Leman ($k$-WL) hierarchy is well understood. However, such architectures often fail to deliver solid predictive performance on real-world tasks, limiting their practical impact. In contrast, global attention-based models such as graph transformers demonstrate strong performance in practice, but comparing their expressive power with the $k$-WL hierarchy remains challenging, particularly since these architectures rely on positional or structural encodings for their expressivity and predictive performance. To address this, we show that the recently proposed Edge Transformer, a global attention model operating on node pairs instead of nodes, has 3-WL expressive power when provided with the right tokenization. Empirically, we demonstrate that the Edge Transformer surpasses other theoretically aligned architectures regarding predictive performance and is competitive with state-of-the-art models on algorithmic reasoning and molecular regression tasks while not relying on positional or structural encodings.

## 1 Introduction

Graph Neural Networks (GNNs) are the de-facto standard in graph learning [16, 43, 28, 50] but suffer from limited expressivity in distinguishing non-isomorphic graphs in terms of the 1-*dimensional Weisfeiler–Leman algorithm* (1-WL) [35, 50]. Hence, recent works introduced *higher-order* GNNs, aligned with the $k$-dimensional Weisfeiler–Leman ($k$-WL) hierarchy for graph isomorphism testing [1, 33, 35, 36, 38], resulting in more expressivity with an increase in $k > 1$. The $k$-WL hierarchy draws from a rich history in graph theory and logic [3, 4, 5, 10, 49], offering a deep theoretical understanding of $k$-WL-aligned GNNs. While theoretically intriguing, higher-order GNNs often fail to deliver state-of-the-art performance on real-world problems, making theoretically grounded models less relevant in practice [1, 36, 38]. In contrast, graph transformers [17, 19, 31, 41, 52] recently demonstrated state-of-the-art empirical performance. However, they draw their expressive power mostly from positional/structural encodings (PEs), making it difficult to understand these models in terms of an expressivity hierarchy such as the $k$-WL. While a few works theoretically aligned graph transformers with the $k$-WL hierarchy [26, 27, 53], we are not aware of any works reporting empirical results for 3-WL-equivalent graph transformers on established graph learning datasets.

In this work, we aim to set the ground for graph learning architectures that are theoretically aligned with the higher-order Weisfeiler–Leman hierarchy while delivering strong empirical performance and, at the same time, demonstrate that such an alignment creates powerful synergies between transformers and graph learning. Hence, we close the gap between theoretical expressivity and real-world predictive power. To this end, we apply the *Edge Transformer* (ET) architecture, initially developed for *systematic generalization* problems [6], to the field of graph learning. Systematic

Mathematics of Modern Machine Learning Workshop at NeurIPS 2024.




(a) Graph $G$ (b) Node pair tensor $\mathbf{X}^{(0)}$

Figure 1: Tokenization of the Edge Transformer. Given a graph $G$, we construct a 3D tensor where we embed information from each node pair into a $d$ dimensional vector.

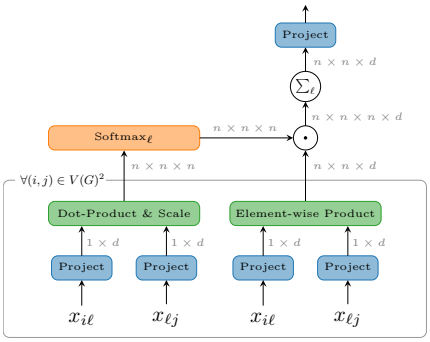



Figure 2: Tensor operations in a single triangular attention head; see Algorithm 1 for a comparison to standard attention in pseudo-code.

(or compositional) generalization refers to the ability of a model to generalize to complex novel concepts by combining primitive concepts observed during training, posing a challenge to even the most advanced models such as GPT-4 [15].

Specifically, we contribute the following:

1. We propose a concrete implementation of the Edge Transformer that readily applies to various graph learning tasks.
2. We show theoretically that this Edge Transformer implementation is as expressive as the 3-WL *without* the need for positional/structural encodings.
3. We demonstrate the benefits of aligning models with the $k$-WL hierarchy by leveraging well-established results from graph theory and logic to develop a theoretical understanding of systematic generalization in terms of first-order logic statements.
4. We demonstrate the superior empirical performance of the resulting architecture compared to a variety of other theoretically motivated models, particularly higher-order GNNs, as well as competitive performance compared to state-of-the-art models in molecular regression and neural algorithmic reasoning tasks.

## 2 Related work

See Appendix A for an extended related work section. Here, we want to highlight PPGN [32], a simple GNN model that uses a series of MLP layers and matrix multiplications to achieve 3-WL expressive power, as well as the Relational Transformer (RT) [12], operating on both nodes and edges and, similar to the ET, builds relational representations, that is, representations on edges. Although the RT integrates edge information into self-attention and hence does not need to resort to positional/structural encodings, the RT is theoretically poorly understood, much like other graph transformers. Other graph transformers with higher-order expressive power are Graphormer-GD [53] and TokenGT [27]. However, Graphormer-GD is strictly less expressive than the 3-WL [53]. Moreover, the graph transformers in Kim et al. [27] are infeasible for higher orders. For example, achieving provably 3-WL expressivity results in a runtime complexity of $O(n^6)$.

Finally, systematic generalization has recently been investigated both empirically and theoretically [6, 15, 25, 42]. In particular, Dziri et al. [15] demonstrate that compositional generalization is lacking in state-of-the-art transformers such as GPT-4.

## 3 Edge Transformers

The ET was originally designed to improve the systematic generalization abilities of machine learning models. To borrow the example from Bergen et al. [6], a model that is presented with relations such as $\text{MOTHER}(x, y)$, indicating that $y$ is the mother of $x$, could generalize to a more complex relation $\text{GRANDMOTHER}(x, z)$, indicating that $z$ is the grandmother of $x$ if $\text{MOTHER}(x, y) \wedge \text{MOTHER}(y, z)$ holds. The particular form of attention used by the ET, which we will formally introduce hereafter, is designed to explicitly model such more complex relations. Indeed, leveraging our theoretical results of Section 4, in Section 5, we formally justify the ET for performing systematic generalization. We will now formally define the ET.

In general, the ET operates on a graph $G$ with nodes $V(G)$ and consecutively updates a 3D tensor state $\boldsymbol{X} \in \mathbb{R}^{n \times n \times d}$, where $d$ is the embedding dimension and $\boldsymbol{X}_{ij}$ or $\boldsymbol{X}(\boldsymbol{u})$ denotes the representation of the node pair $\boldsymbol{u} \coloneqq (i, j) \in V(G)^2$; see Figure 1 for a visualization of this construction. Concretely, the $t$-th ET layer computes

$$\boldsymbol{X}_{ij}^{(t)} \coloneqq \mathsf{FFN}\big(\boldsymbol{X}_{ij}^{(t-1)} + \mathsf{TriAttention}\big(\mathsf{LN}\big(\boldsymbol{X}_{ij}^{(t-1)}\big)\big)\big),$$

for each node pair $(i, j)$, where $\mathsf{FFN}$ is a feed-forward neural network, $\mathsf{LN}$ denotes layer normalization [2] and $\mathsf{TriAttention}$ is defined as

$$\mathsf{TriAttention}(\boldsymbol{X}_{ij}) \coloneqq \sum_{l=1}^{n} \alpha_{ilj} \boldsymbol{V}_{ilj}, \tag{1}$$

which computes a tensor product between a three-dimensional *attention tensor* $\alpha$ and a three-dimensional *value tensor* $\mathbf{V}$, by multiplying and summing over the second dimension. Here,

$$\alpha_{ilj} \coloneqq \underset{l \in [n]}{\mathsf{softmax}}\Big(\frac{1}{\sqrt{d}} \boldsymbol{X}_{il} \boldsymbol{W}^{Q} \big(\boldsymbol{X}_{lj} \boldsymbol{W}^{K}\big)^{T}\Big) \in \mathbb{R} \tag{2}$$

is the attention score between the features of tuples $(i, l)$ and $(l, j)$, and

$$\boldsymbol{V}_{ilj} \coloneqq \boldsymbol{X}_{il} \boldsymbol{W}^{V_1} \odot \boldsymbol{X}_{lj} \boldsymbol{W}^{V_2}, \tag{3}$$

we call *value fusion* of the tuples $(i, l)$ and $(l, j)$ with $\odot$ denoting element-wise multiplication. Moreover, $\boldsymbol{W}^{Q}, \boldsymbol{W}^{K}, \boldsymbol{W}^{V_1}, \boldsymbol{W}^{V_2} \in \mathbb{R}^{d \times d}$ are learnable projection matrices; see Figure 2 for an overview of the tensor operations in triangular attention and see Algorithm 1 for a comparison to standard attention [45] in pseudo-code. Note that similar to standard attention, triangular attention can be straightforwardly extended to multiple heads.

As we will show in Section 4, the ET owes its expressive power to the special form of triangular attention. In our implementation of the ET, we use the following tokenization, which is sufficient to obtain our theoretical result.

**Tokenization** Let $G \coloneqq (V(G), E(G), \ell)$ be a graph with $n$ nodes, feature matrix $\boldsymbol{F} \in \mathbb{R}^{n \times p}$ and edge feature tensor $\boldsymbol{E} \in \mathbb{R}^{n \times n \times q}$. If no edge features are available, we randomly initialize learnable vectors $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^{q}$ and assign $\boldsymbol{x}_1$ to $\boldsymbol{E}_{ij}$ if $(i, j) \in E(G)$. Further, for all $i \in V(G)$, we assign $\boldsymbol{x}_2$ to $\boldsymbol{E}_{ii}$. Lastly, if $(i, j) \notin E(G)$ and $i \neq j$, we set $\boldsymbol{E}_{ij} = \mathbf{0}$. We then construct a 3D tensor of input tokens $\boldsymbol{X} \in \mathbb{R}^{n \times n \times d}$, such that for node pair $(i, j) \in V(G)^2$,

$$\boldsymbol{X}_{ij} \coloneqq \phi\big([\boldsymbol{E}_{ij} \quad \boldsymbol{F}_i \quad \boldsymbol{F}_j]\big) \in \mathbb{R}^{d}, \tag{4}$$

where $\phi \colon \mathbb{R}^{2p+q} \to \mathbb{R}^{d}$ is a neural network. Extending Bergen et al. [6], our tokenization additionally considers node features, making it more appropriate for the graph learning setting.

**Efficiency** The triangular attention above imposes a $\mathcal{O}(n^3)$ runtime and memory complexity, which is significantly more efficient than other transformers with 3-$\mathsf{WL}$ expressive power, such as the higher-order transformers in Kim et al. [26] and Kim et al. [27] with a runtime of $\mathcal{O}(n^6)$. Nonetheless, the ET is still significantly less efficient than most graph transformers, with a runtime of $\mathcal{O}(n^2)$ [31, 41, 52]. Thus, the ET is currently only applicable to mid-sized graphs; see Appendix G for an extended discussion of this limitation.

**Positional/structural encodings** Additionally, GNNs and graph transformers often benefit empirically from added positional/structural encodings [13, 31, 41]. We can easily add PEs to the above tokens with the ET. Specifically, we can encode any PEs for node $i \in V(G)$ as an edge feature in $\boldsymbol{E}_{ii}$ and any PEs between a node pair $(i, j) \in V(G)^2$ as an edge feature in $\boldsymbol{E}_{ij}$. Note that typically, PEs between pairs of nodes are incorporated during the attention computation of graph transformers [31, 52]. However, in Section 6, we demonstrate that simply adding these PEs to our tokens is also viable for improving the empirical results of the ET.

With tokenization as defined above, the ET can now be used on many graph learning problems, encoding both node and edge features and making predictions for node pair-, edge-, node-, and graph-level tasks; see Appendix D for additional implementation details. We refer to a concrete set of parameters of the ET, including tokenization and positional/structural encodings, as a *parameterization*. We now move on to our theoretical result, showing that the ET has an expressive power of at least the 3-WL.

## 4 The Expressivity of Edge Transformers

Here, we relate the ET to the *folklore* Weisfeiler–Leman ($k$-FWL) hierarchy, a variant of the $k$-WL hierarchy for which, for $k > 2$, $(k-1)$-FWL is as expressive as $k$-WL [18]. Specifically, we show that the ET can simulate the 2-FWL, resulting in 3-WL expressive power. For detailed background on the $k$-WL and $k$-FWL hierarchy, see Appendix H. Concretely, we obtain the following theorem; see Appendix J for a formal statement and proof details.

**Theorem 1** (Informal)**.** *The ET has exactly* 3-*WL expressive power.*

Note that following previous works [32, 36, 38], our expressivity result is *non-uniform* in that our result only holds for an arbitrary but fixed graph size $n$; see Proposition 7 and Proposition 8 for the complete formal statements and proof of Theorem 1; see Appendix I for some intuition on how the ET can simulate the 2-FWL.

Interestingly, our proofs do not resort to positional/structural encodings. The ET draws its 3-WL expressive power from its aggregation scheme, the triangular attention. In Section 6, we demonstrate that this also holds in practice, where the ET performs strongly without additional encodings. In what follows, we use the above results to derive a more principled understanding of the ET in terms of systematic generalization, for which it was originally designed. Thereby, we demonstrate that graph theoretic results can also be leveraged in other areas of machine learning, further highlighting the benefits of theoretically grounded models.

## 5 The Logic of Edge Transformers

After borrowing the ET from systematic generalization in the previous section, we now return the favor. Specifically, we use a well-known connection between graph isomorphism and first-order logic to obtain a theoretical justification for systematic generalization reasoning using the ET. Here, we give an informal overview over the results. For a detailed and formal discussion; see Appendix C. Recalling the example around the GRANDMOTHER relation composed from the more primitive MOTHER relation in Section 3, Bergen et al. [6] go ahead and argue that since self-attention of standard transformers is defined between pairs of nodes, learning explicit representations of GRANDMOTHER is impossible and that learning such representations implicitly incurs a high burden on the learner. Conversely, the authors argue that since the Edge Transformer computes triangular attention over triplets of nodes and computes explicit representations between node pairs, the Edge Transformer can systematically generalize to relations such as GRANDMOTHER. While Bergen et al. [6] argue the above intuitively, we will now utilize the connection between first-order logic (FO-logic) and graph isomorphism established in Cai et al. [10] to develop a theoretical understanding of systematic generalization; see Appendix H for an introduction to first-order logic over graphs.

Concretely, the results in Cai et al. [10] establish a correspondence between $k$-FWL expressivity and the ability to evaluate first-order logic statements with counting quantifiers and $k+1$ variables. Moreover, the number iterations performed by the $k$-FWL correspond to the maximum number of nested quantifiers of statements that can be evaluated.

Table 1: Average test results and standard deviation for the molecular regression datasets over five random seeds.

| Model | ZINC (12K) | ALCHEMY (12K) |
|---|---|---|
| | MAE ↓ | MAE ↓ |
| CIN [8] | 0.079 ±0.006 | – |
| Graphormer-GD [53] | 0.081 ±0.009 | – |
| SignNet [29] | 0.084 ±0.006 | 0.113 ±0.002 |
| BasisNet [21] | 0.155 ±0.007 | 0.110 ±0.001 |
| PPGN++ [40] | 0.071 ±0.001 | 0.109 ±0.001 |
| SPE [21] | **0.069 ±0.004** | **0.108 ±0.001** |
| ET | **0.062 ±0.004** | **0.099 ± 0.001** |
| ET+RRWP | **0.059 ±0.004** | **0.098 ± 0.001** |

Table 2: ZINC (12K) leaderboard.

| Model | ZINC (12K) |
|---|---|
| | MAE ↓ |
| Graphormer-GD [53] | 0.081 ±0.009 |
| CIN [8] | 0.079 ±0.006 |
| Graph-MLP-Mixer [19] | 0.073 ±0.001 |
| PPGN++ [40] | 0.071 ±0.001 |
| GraphGPS [41] | 0.070 ±0.004 |
| SPE [21] | 0.069 ±0.004 |
| Graph Diffuser [17] | 0.068 ±0.002 |
| Specformer [7] | 0.066 ±0.003 |
| GRIT [31] | **0.059 ±0.002** |
| ET | **0.062 ±0.004** |
| ET+RRWP | **0.059 ±0.004** |

This correspondence lets us, for example, say that the ET, with its 2-FWL expressive power, can evaluate the example given in Bergen et al. [6], namely,

$$\text{GRANDMOTHER}(x, z) = \exists y \big(\text{MOTHER}(x, y) \wedge \text{MOTHER}(y, z)\big).$$

Here, the grandmother relation is described via a first-order logic statement with 1 quantifier and 3 variables.

More generally, our results in Section 4, combined with the results in Cai et al. [10], result in a theoretical justification for the intuitive argument made by Bergen et al. [6], namely that the ET can learn an *explicit* representation of a novel concept, in our example the GRANDMOTHER relation. Moreover, note that the GRANDMOTHER relation can be evaluated in a single iteration and is a relation over 2 variables. As a result, two iterations of the 2-FWL allow us to evaluate the statement

$$\text{GREATGRANDMOTHER}(x, a) = \exists y \big(\text{GRANDMOTHER}(x, y) \wedge \text{MOTHER}(y, a)\big),$$

where GRANDMOTHER is a generalized concept obtained from the primitive concept MOTHER and GREATGRANDMOTHER is generalized from GRANDMOTHER and MOTHER and can be described with 3 variables and two nested quantifiers; see Appendix C for a more formal treatment of this connection between first-order logic and the 2-FWL.

To summarize, knowing the expressive power of a model such as the ET in terms of the Weisfeiler–Leman hierarchy allows us to draw direct connections to the logical reasoning abilities of the model. Further, this theoretical connection allows an interpretation of systematic generalization as the ability of a model with the expressive power of at least the $k$-FWL to iteratively re-combine concepts from first principles (such as the MOTHER relation) as a hierarchy of first-order logic statements with counting quantifiers and at most $k + 1$ variables.

## 6 Experimental evaluation

Here, we investigate how well the ET performs on various graph-learning tasks. We include tasks on graph-, node-, and edge-level. Specifically, we answer the following questions.

**Q1** How does the ET fare against other theoretically aligned architectures regarding predictive performance?

**Q2** How does the ET compare to state-of-the-art models?

**Q3** How effectively can the ET benefit from additional positional/structural encodings?

We provide the full experimental evaluation in Appendix E. Here, we want to highlight a few results and answer the questions posed above. In our tables, we highlight **first**, **second** and **third** best results.

### 6.1 How does the ET fare against other theoretically aligned architectures regarding predictive performance?

In Table 1, we present results on two popular molecular property prediction tasks, namely ZINC (12K) [14] and ALCHEMY (12K), comparing the ET to six theoretically grounded graph models. We

Table 3: Average test micro F1 of different algorithm classes and average test score of all algorithms in CLRS over ten random seeds; see Appendix E.4 for test scores per algorithm and Appendix E.5 for details on the standard deviation.

| **Model** | Sorting | Searching | DC | Greedy | DP | Graphs | Strings | Geometry | *Average* | *All algorithms* |
|---|---|---|---|---|---|---|---|---|---|---|
| Deep Sets [12] | 68.89 | 50.99 | 12.29 | 77.83 | 68.29 | 42.09 | 2.92 | 65.47 | 48.60 | 50.29 |
| GAT [12] | 21.25 | 38.04 | 15.19 | 75.75 | 63.88 | 55.53 | 1.57 | 68.94 | 41.82 | 48.08 |
| MPNN [12] | 27.12 | 43.94 | 16.14 | 89.40 | 68.81 | 63.30 | 2.09 | 83.03 | 49.23 | 55.15 |
| PGN [12] | 28.93 | 60.39 | 51.30 | 76.72 | 71.13 | 64.59 | 1.82 | 67.78 | 52.83 | 56.57 |
| RT [12] | 50.01 | 65.31 | 66.52 | 85.32 | 83.20 | 65.33 | 32.52 | 84.55 | 66.60 | 66.18 |
| Triplet-GMPNN [23] | 60.37 | 58.61 | 76.36 | 91.21 | 81.99 | 81.41 | 49.09 | 94.09 | 74.14 | 75.98 |
| ET | 82.26 | 63.00 | 64.44 | 81.67 | 83.49 | 86.08 | 54.84 | 88.22 | 75.51 | 80.13 |

evaluate the ET with and without the positional/structural encodings RRWP, proposed in Ma et al. [31]; see Section 6.3 for a discussion on the impact of these encodings on the performance of the ET. In summary, we find that the ET outperforms other theoretically grounded models, even without positional/structural encodings; see Table 1 for an extension of Table 1 with more baselines and one more molecular property prediction task.

### 6.2 How does the ET compare to state-of-the-art models?

In Table 2, we provide a comparison of the ET with the best overall models on ZINC (12K). Here, we find that the ET is highly competitive with the best models, even without using positional/structural encodings.

Further, we want to evaluate the performance of the ET on another domain than molecular property prediction. To this end, in Table 3, we evaluate the ET on the CLRS benchmark for neural algorithmic reasoning [46]. Here, the input, output, and intermediate steps of 30 classical algorithms are translated into graph data, where nodes represent the algorithm input and edges are used to encode a partial ordering of the input. The algorithms are divided into 8 algorithm classes, such as sorting, dynamic programming or divide and conquer. Of independent interest to algorithmic reasoning is the fact that the CLRS benchmark requires predictions on both node- and edge-level, which serves as a challenging setting for graph models. We evaluate on two strong baselines on this benchmark and stick closely to their training and evaluation setup. We find that the ET is a highly competitive model on this benchmark. In particular, the ET has the highest average performance over all algorithm classes and all algorithms.

### 6.3 How effectively can the ET benefit from additional positional/structural encodings?

To determine the impact of positional/structural encodings, we point again to our results in Table 1. Here, we use RRWP, the positional/structural encodings proposed for GRIT [31], the best model on ZINC (12K). We find that, while the use of positional/structural encodings has a positive impact on performance, the ET does not depend on such encodings for strong performance, even when compared the best models; see Table 2. In comparison, GRIT is highly reliant on RRWP encodings; see Table 5 in Ma et al. [31].

## 7 Conclusion

Here, we established a previously unknown connection between the Edge Transformer and the 3-WL and enabled the Edge Transformer for various graph learning tasks, including graph-, node-, and edge-level tasks. We also utilized a well-known connection between graph isomorphism testing and first-order logic to derive a theoretical interpretation of systematic generalization. We demonstrated empirically that the Edge Transformer is a promising architecture for graph learning, outperforming other theoretically aligned architectures and being among the best models on ZINC (12K) and CLRS. Furthermore, the ET is a graph transformer that does not rely on positional/structural encodings for strong empirical performance. Future work could further explore the potential of the Edge Transformer in neural algorithmic reasoning and molecular learning by improving its scalability to larger graphs, in particular through architecture-specific low-level GPU optimizations and model parallelism.

## References


[1] W. Azizian and M. Lelarge. Characterizing the expressive power of invariant and equivariant graph neural networks. In *International Conference on Learning Representations*, 2021.

[2] L. J. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *Arxiv preprint*, 2016.

[3] L. Babai. Lectures on graph isomorphism. University of Toronto, Department of Computer Science. Mimeographed lecture notes, October 1979, 1979.

[4] L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *Symposium on Foundations of Computer Science*, pages 39–46, 1979.

[5] L. Babai, P. Erdős, and S. M. Selkow. Random graph isomorphism. *SIAM Journal on Computing*, pages 628–635, 1980.

[6] L. Bergen, T. J. O'Donnell, and D. Bahdanau. Systematic generalization with edge transformers. In *Advances in Neural Information Processing Systems*, 2021.

[7] D. Bo, C. Shi, L. Wang, and R. Liao. Specformer: Spectral graph neural networks meet transformers. In *International Conference on Learning Representations*, 2023.

[8] C. Bodnar, F. Frasca, N. Otter, Y. G. Wang, P. Liò, G. Montúfar, and M. M. Bronstein. Weisfeiler and Lehman go cellular: CW networks. In *Advances in Neural Information Processing Systems*, 2021.

[9] M. Bohde, M. Liu, A. Saxton, and S. Ji. On the markov property of neural algorithmic reasoning: Analyses and methods. In *International Conference on Learning Representations*, 2024.

[10] J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992.

[11] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, 2022.

[12] C. Diao and R. Loynd. Relational attention: Generalizing transformers for graph-structured tasks. In *International Conference on Learning Representations*, 2023.

[13] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2022.

[14] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24, 2023.

[15] N. Dziri, X. Lu, M. Sclar, X. L. Li, L. Jiang, B. Y. Lin, S. Welleck, P. West, C. Bhagavatula, R. L. Bras, J. D. Hwang, S. Sanyal, X. Ren, A. Ettinger, Z. Harchaoui, and Y. Choi. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems*, 2023.

[16] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017.

[17] D. Glickman and E. Yahav. Diffusing graph attention. *ArXiv preprint*, 2023.

[18] M. Grohe. The logic of graph neural networks. In *Symposium on Logic in Computer Science*, pages 1–17, 2021.

[19] X. He, B. Hooi, T. Laurent, A. Perold, Y. LeCun, and X. Bresson. A generalization of vit/mlp-mixer to graphs. In *International Conference on Machine Learning*, 2023.

[20] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec. OGB-LSC: A large-scale challenge for machine learning on graphs. In *NeurIPS: Datasets and Benchmarks Track*, 2021.

[21] Y. Huang, W. Lu, J. Robinson, Y. Yang, M. Zhang, S. Jegelka, and P. Li. On the stability of expressive positional encodings for graph neural networks. *Arxiv preprint*, 2023.

[22] M. S. Hussain, M. J. Zaki, and D. Subramanian. Global self-attention as a replacement for graph convolution. In A. Zhang and H. Rangwala, editors, *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 655–665, 2022.

[23] B. Ibarz, V. Kurin, G. Papamakarios, K. Nikiforou, M. Bennani, R. Csordás, A. J. Dudzik, M. Bosnjak, A. Vitvitskyi, Y. Rubanova, A. Deac, B. Bevilacqua, Y. Ganin, C. Blundell, and P. Velickovic. A generalist neural algorithmic learner. In *Learning on Graphs Conference*, 2022.

[24] Y. Jung and S. Ahn. Triplet edge attention for algorithmic reasoning. *Arxiv preprint*, 2023.

[25] D. Keysers, N. Schärli, N. Scales, H. Buisman, D. Furrer, S. Kashubin, N. Momchev, D. Sinopalnikov, L. Stafiniak, T. Tihon, D. Tsarkov, X. Wang, M. van Zee, and O. Bousquet. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*, 2020.

[26] J. Kim, S. Oh, and S. Hong. Transformers generalize deepsets and can be extended to graphs & hypergraphs. In *Advances in Neural Information Processing Systems*, 2021.

[27] J. Kim, T. D. Nguyen, S. Min, S. Cho, M. Lee, H. Lee, and S. Hong. Pure transformers are powerful graph learners. *ArXiv preprint*, 2022.

[28] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[29] D. Lim, J. Robinson, L. Zhao, T. E. Smidt, S. Sra, H. Maron, and S. Jegelka. Sign and basis invariant networks for spectral graph representation learning. *ArXiv preprint*, 2022.

[30] Y. Lipman, O. Puny, and H. Ben-Hamu. Global attention improves graph networks generalization. *ArXiv preprint*, 2020.

[31] L. Ma, C. Lin, D. Lim, A. Romero-Soriano, K. Dokania, M. Coates, P. H.S. Torr, and S.-N. Lim. Graph Inductive Biases in Transformers without Message Passing. In *International Conference on Machine Learning*, 2023.

[32] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, 2019.

[33] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019.

[34] H. Maron, E. Fetaya, N. Segol, and Y. Lipman. On the universality of invariant networks. In *International Conference on Machine Learning*, pages 4363–4371, 2019.

[35] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence*, 2019.

[36] C. Morris, G. Rattan, and P. Mutzel. Weisfeiler and Leman go sparse: Towards higher-order graph embeddings. In *Advances in Neural Information Processing Systems*, 2020.

[37] C. Morris, Y. L., H. Maron, B. Rieck, N. M. Kriege, M. Grohe, M. Fey, and K. Borgwardt. Weisfeiler and Leman go machine learning: The story so far. *ArXiv preprint*, 2021.

[38] C. Morris, G. Rattan, S. Kiefer, and S. Ravanbakhsh. SpeqNets: Sparsity-aware permutation-equivariant graph networks. In *International Conference on Machine Learning*, pages 16017–16042, 2022.

[39] L. Müller, M. Galkin, C. Morris, and L. Rampásek. Attending to graph transformers. *Transactions on Machine Learning Research*, 2024.

[40] O. Puny, D. Lim, B. T. Kiani, H. Maron, and Y. Lipman. Equivariant polynomials for graph neural networks. *ArXiv preprint*, 2023.

[41] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 2022.

[42] Y. Ren, S. Lavoie, M. Galkin, D. J. Sutherland, and A. Courville. Improving compositional generalization using iterated learning and simplicial embeddings. In *Advances in Neural Information Processing Systems*, 2023.

[43] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 2009.

[44] P. Tillet, H. Kung, and D. D. Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2019.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[46] P. Velickovic, A. P. Badia, D. Budden, R. Pascanu, A. Banino, M. Dashevskiy, R. Hadsell, and C. Blundell. The CLRS algorithmic reasoning benchmark. In *International Conference on Machine Learning*, 2022.

[47] O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. In *International Conference on Learning Representations*, 2016.

[48] Y. Wang and M. Zhang. Towards better evaluation of GNN expressiveness with BREC dataset. *Arxiv preprint*, 2023.

[49] B. Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

[50] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[51] K. Xu, L. Wang, M. Yu, Y. Feng, Y. Song, Z. Wang, and D. Yu. Cross-lingual knowledge graph alignment via graph matching neural network. In *Annual Meeting of the Association for Computational Linguistics*, 2019.

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

[53] B. Zhang, S. Luo, L. Wang, and D. He. Rethinking the expressive power of gnns via graph biconnectivity. In *International Conference on Learning Representations*, 2023.

## A Extended related work

Many graph learning models with higher-order WL expressive power exist, notably $\delta$-$k$-GNNs [36], SpeqNets [38], $k$-IGNs [34, 33], PPGN [32], and the more recent PPGN++ [40]. Moreover, Lipman et al. [30] devised a low-rank attention module possessing the same power as the folklore 2-WL and Bodnar et al. [8] proposed CIN with an expressive power of at least 3-WL. For an overview of Weisfeiler–Leman in graph learning, see Morris et al. [37].

Many graph transformers exist, notably Graphormer [52] and GraphGPS [41]. However, state-of-the-art graph transformers typically rely on positional/structural encodings, which makes it challenging to derive a theoretical understanding of their expressive power. Graph transformers with higher-order expressive power are Graphormer-GD [53] and TokenGT [27] as well as the higher-order graph transformers in Kim et al. [26]. However, Graphormer-GD is strictly less expressive than the 3-WL [53]. Further, Kim et al. [26] and Kim et al. [27] align transformers with $k$-IGNs and, thus, obtain the theoretical expressive power of the corresponding $k$-WL but do not empirically evaluate their transformers for $k > 2$. In addition, these higher-order transformers suffer from a $\mathcal{O}(n^{2k})$ runtime and memory complexity. For comparison, with $k = 3$, the ET offers provable 3-WL expressivity with $\mathcal{O}(n^3)$ runtime and memory complexity, several orders of magnitude more efficient than the corresponding 3-WL expressive transformer in Kim et al. [27]. For an overview of graph transformers, see Müller et al. [39].

## B Comparison of TriAttention to standard attention

Algorithm 1 shows a comparison between TriAttention and standard attention [45] in pseudo-code. Since TriAttention operates on pairs of tokens, query-, key- and value- operations are lifted to tensor operations. In addition, TriAttention uses value fusion to construct the value tensor; see Section 3.

**Algorithm 1** Comparison between standard attention and triangular attention in PyTorch-like pseudo-code.

**function** ATTENTION($\mathbf{X} : n \times d$)
  $\mathbf{Q}, \mathbf{K}, \mathbf{V} \leftarrow \mathsf{linear}(\mathbf{X}).\mathsf{chunk}(3)$
  # no op
  $\tilde{\mathbf{A}} \leftarrow \mathsf{einsum}(id, jd \to ij, \mathbf{Q}, \mathbf{K})$
  $\mathbf{A} \leftarrow \mathsf{softmax}(\tilde{\mathbf{A}}/\sqrt{d}, -1)$
  $\mathbf{O} \leftarrow \mathsf{einsum}(ij, jd \to id, \mathbf{A}, \mathbf{V})$
  **return** $\mathsf{linear}(\mathbf{O})$
**end function**

**function** TRI_ATTENTION($\mathbf{X} : n \times n \times d$)
  $\mathbf{Q}, \mathbf{K}, \mathbf{V}^1, \mathbf{V}^2 \leftarrow \mathsf{linear}(\mathbf{X}).\mathsf{chunk}(4)$
  $\mathbf{V} \leftarrow \mathsf{einsum}(ild, ljd \to iljd, \mathbf{V}^1, \mathbf{V}^2)$
  $\tilde{\mathbf{A}} \leftarrow \mathsf{einsum}(ild, ljd \to ilj, \mathbf{Q}, \mathbf{K})$
  $\mathbf{A} \leftarrow \mathsf{softmax}(\tilde{\mathbf{A}}/\sqrt{d}, -1)$
  $\mathbf{O} \leftarrow \mathsf{einsum}(ilj, iljd \to ijd, \mathbf{A}, \mathbf{V})$
  **return** $\mathsf{linear}(\mathbf{O})$
**end function**

## C Extended: The Logic of Edge Transformers

Here, we use a well-known connection between graph isomorphism and first-order logic to obtain a theoretical justification for systematic generalization reasoning using the ET by leveraging the results in Cai et al. [10]. We will now briefly introduce the most important concepts in Cai et al. [10] and then relate them to systematic generalization of the ET and similar models.

**Language and configurations** Here, we consider FO-logic statements with counting quantifiers and denote with $\mathcal{C}_{k,m}$ the language of all such statements with at most $k$ variables and quantifier depth $m$. A *configuration* is a map between first-order variables and nodes in a graph. Concretely, configurations let us define a statement $\varphi$ in first-order logic, such as three nodes forming a triangle, without speaking about concrete nodes in a graph $G = (V(G), E(G))$. Instead, we can use a configuration to map the three variables in $\varphi$ to nodes $v, w, u \in V(G)$ and evaluate $\varphi$ to determine whether $v, w$ and $u$ form a triangle in $G$. Of particular importance to us are $k$-configurations $f$ where we map $k$ variables $x_1, \ldots, x_k$ in a FO-logic statement to a $k$-tuple $\boldsymbol{u} \in V(G)^k$ such that $\boldsymbol{u} = (f(x_1), \ldots, f(x_k))$. This lets us now state the following result in Cai et al. [10], relating FO-logic satisfiability to the $k$-FWL hierarchy.

**Theorem 2** (Theorem 5.2 [10], informally)**.** *Let $G \coloneqq (V(G), E(G))$ and $H \coloneqq (V(H), E(H))$ be two graphs with $n$ nodes and let $k \geq 1$. Let $f$ be a $k$-configuration mapping to tuple $\boldsymbol{u} \in V(G)^k$ and let $g$ be a $k$-configuration mapping to tuple $\boldsymbol{v} \in V(H)^k$. Then, for every $t \geq 0$,*

$$C_t^{k,F}(\boldsymbol{u}) = C_t^{k,F}(\boldsymbol{v}),$$

*if, and only, if $\boldsymbol{u}$ and $\boldsymbol{v}$ satisfy the same sentences in $\mathcal{C}_{k+1,t}$.*

Together with Theorem 1, we obtain the above results also for the embeddings of the ET for $k = 2$.

**Corollary 3.** *Let $G \coloneqq (V(G), E(G))$ and $H \coloneqq (V(H), E(H))$ be two graphs with $n$ nodes and let $k = 2$. Let $f$ be a 2-configuration mapping to node pair $\boldsymbol{u} \in V(G)^2$ and let $g$ be a 2-configuration mapping to node pair $\boldsymbol{v} \in V(H)^k$. Then, for every $t \geq 0$,*

$$\boldsymbol{X}^{(t)}(\boldsymbol{u}) = \boldsymbol{X}^{(t)}(\boldsymbol{v}),$$

*if, and only, if $\boldsymbol{u}$ and $\boldsymbol{v}$ satisfy the same sentences in $\mathcal{C}_{3,t}$.*

**Systematic generalization** Returning to the example in Bergen et al. [6], the above result tells us that a model with 2-FWL expressive power and at least $t$ layers is equivalently able to evaluate sentences in $\mathcal{C}_{3,t}$, including

$$\text{GRANDMOTHER}(x, z) = \exists y \big(\text{MOTHER}(x, y) \wedge \text{MOTHER}(y, z)\big),$$

i.e., the grandmother relation, and store this information encoded in some 2-tuple representation $\boldsymbol{X}^{(t)}(\boldsymbol{u})$, where $\boldsymbol{u} = (u, v)$ and $u$ is a grandmother of $v$. As a result, we have theoretical justification for the intuitive argument made by Bergen et al. [6], namely that the ET can learn an *explicit* representation of a novel concept, in our example the GRANDMOTHER relation.

However, when closely examining the language $\mathcal{C}_{3,t}$, we find that the above result allows for an even wider theoretical justification of the systematic generalization ability of the ET. Concretely, we will show that once the ET obtains a representation for a novel concept such as the GRANDMOTHER relation, at some layer $t$, the ET can re-combine said concept to generalize to even more complex concepts. For example, consider the following relation, which we naively write as

$$\text{GREATGRANDMOTHER}(x, a) = \exists z \exists y \big(\text{MOTHER}(x, y) \wedge \text{MOTHER}(y, z) \wedge \text{MOTHER}(z, a)\big).$$

At first glance, it seems as though GREATGRANDMOTHER $\in \mathcal{C}_{4,1}$ but GREATGRANDMOTHER $\notin \mathcal{C}_{3,t}$ for any $t \geq 1$. However, notice that the variable $y$ serves merely as an intermediary to establish the GRANDMOTHER relation. Hence, we can, without loss of generality, write the above as

$$\text{GREATGRANDMOTHER}(x, a) = \exists y \underbrace{\big(\exists a \big(\text{MOTHER}(x, a) \wedge \text{MOTHER}(a, y)\big)\big)}_{a \text{ is re-quantified and temporarily bound}} \wedge \text{MOTHER}(y, a)\big),$$

i.e., we *re-quantify* $a$ to temporarily serve as the mother of $x$ and the daughter of $y$. Afterwards, $a$ is released and again refers to the great grandmother of $x$. As a result, GREATGRANDMOTHER $\in \mathcal{C}_{3,2}$ and hence the ET, as well as any other model with at least 2-FWL expressive power, is able to generalize to the GREATGRANDMOTHER relation within two layers, by iteratively re-combining existing concepts, in our example the GRANDMOTHER and the MOTHER relation. This becomes even more clear, by writing

$$\text{GREATGRANDMOTHER}(x, a) = \exists y \big(\text{GRANDMOTHER}(x, y) \wedge \text{MOTHER}(y, a)\big),$$

where GRANDMOTHER is a generalized concept obtained from the primitive concept MOTHER. To summarize, knowing the expressive power of a model such as the ET in terms of the Weisfeiler–Leman hierarchy allows us to draw direct connections to the logical reasoning abilities of the model. Further, this theoretical connection allows an interpretation of systematic generalization as the ability of a model with the expressive power of at least the $k$-FWL to iteratively re-combine concepts from first principles (such as the MOTHER relation) as a hierarchy of statements in $\mathcal{C}_{k+1,t}$, containing all FO-logic statements with counting quantifiers, at most $k + 1$ variables and quantifier depth $t$.

## D Implementation details

Since the Edge Transformer already builds representations on node pairs, making predictions for node pair- or edge-level tasks is straightforward. Specifically, let $L$ denote the number of Edge Transformer layers. Then, for a node pair $(i, j) \in V(G)^2$, we simply readout $\boldsymbol{X}_{ij}^{(L)}$, where on the edge-level we restrict ourselves to the case where $(i, j) \in E(G)$. In what follows, we propose a pooling method from node pairs to nodes, which allows us also to make predictions for node- and graph-level tasks. For each node $i \in V(G)$, we compute

$$\mathsf{ReadOut}(i) \coloneqq \sum_{j \in [n]} \rho_1(\boldsymbol{X}_{ij}^{(L)}) + \rho_2(\boldsymbol{X}_{ji}^{(L)}),$$

where $\rho_1, \rho_2$ are neural networks. We apply $\rho_1$ to node pairs where node $i$ is at the first position and $\rho_2$ to node pairs where node $i$ is at the second. We found that making such a distinction has positive impacts on empirical performance. Then, for graph-level predictions, we first compute node-level readout as above and then use common graph-level pooling functions such as `sum` and `mean` [50] or `set2seq` [47] on the resulting node representations.

## E Experimental results

Here, we investigate how well the ET performs on various graph-learning tasks. We include tasks on graph-, node-, and edge-level. Specifically, we answer the following questions.

**Q1** How does the ET fare against other theoretically aligned architectures regarding predictive performance?

**Q2** How does the ET compare to state-of-the-art models?

**Q3** How effectively can the ET benefit from additional positional/structural encodings?

The source code for our experiments is available at `https://github.com/luis-mueller/towards-principled-gts`. To foster research in principled graph transformers such as the ET, we provide accessible implementations of ET, both in PyTorch and Jax.

**Datasets** We evaluate the ET on graph-, node-, and edge-level tasks from various domains to demonstrate its versatility.

On the graph level, we evaluate the ET on the molecular datasets ZINC (12K), ZINC-FULL [14], ALCHEMY (12K), and PCQM4MV2 [20]. Here, nodes represent atoms and edges bonds between atoms, and the task is always to predict one or more molecular properties of a given molecule. Due to their relatively small graphs, the above datasets are ideal for evaluating higher-order and other resource-hungry models.

On the node and edge level, we evaluate the ET on the CLRS benchmark for neural algorithmic reasoning [46]. Here, the input, output, and intermediate steps of 30 classical algorithms are translated into graph data, where nodes represent the algorithm input and edges are used to encode a partial ordering of the input. The algorithms of CLRS are typically grouped into eight algorithm classes: Sorting, Searching, Divide and Conquer, Greedy, Dynamic Programming, Graphs, Strings, and Geometry. The task is then to predict the output of an algorithm given its input. This prediction is made based on an encoder-processor-decoder framework introduced by Velickovic et al. [46], which is recursively applied to execute the algorithmic steps iteratively. We will use the ET as the processor in this framework, receiving as input the current algorithmic state in the form of node and edge features and outputting the updated node and edge features, according to the latest version of CLRS, available at `https://github.com/google-deepmind/clrs`. As such, the CLRS requires the ET to make both node- and edge-level predictions.

Finally, we conduct empirical expressivity tests on the BREC benchmark [48]. BREC contains 400 pairs of non-isomorphic graphs with up to 198 nodes, ranging from basic, 1-$\mathsf{WL}$ distinguishable graphs to graphs even indistinguishable by 4-$\mathsf{WL}$. In addition, BREC comes with its own training and evaluation pipeline. Let $f \colon \mathcal{G} \to \mathbb{R}^d$ be the model whose expressivity we want to test, where $f$ maps from a set of graphs $\mathcal{G}$ to $\mathbb{R}^d$ for some $d > 0$. Let $(G, H)$ be a pair of non-isomorphic graphs.

Table 4: Average test results and standard deviation for the molecular regression datasets. ALCHEMY (12K) and ZINC-FULL over 5 random seeds, ZINC (12K) over 10 random seeds.

| Model | ZINC (12K) | ALCHEMY (12K) | ZINC-FULL |
|---|---|---|---|
| | MAE ↓ | MAE ↓ | MAE ↓ |
| GIN(E) [50, 40] | 0.163 ±0.03 | 0.180 ±0.006 | 0.180 ±0.006 |
| CIN [8] | 0.079 ±0.006 | – | **0.022 ±0.002** |
| Graphormer-GD [53] | 0.081 ±0.009 | – | 0.025 ±0.004 |
| SignNet [29] | 0.084 ±0.006 | 0.113 ±0.002 | **0.024 ±0.003** |
| BasisNet [21] | 0.155 ±0.007 | 0.110 ±0.001 | – |
| PPGN++ [40] | 0.071 ±0.001 | 0.109 ±0.001 | **0.020 ±0.001** |
| SPE [21] | **0.069 ±0.004** | **0.108 ±0.001** | – |
| ET | **0.062 ±0.004** | **0.099 ± 0.001** | 0.026 ±0.003 |
| ET+RRWP | **0.059 ±0.004** | **0.098 ± 0.001** | **0.024 ±0.003** |

During training, $f$ is trained to maximize the cosine distance between graph embeddings $f(G)$ and $f(H)$. During the evaluation, BREC decides whether $f$ can distinguish $G$ and $H$ by conducting a Hotelling's T-square test with the null hypothesis that $f$ cannot distinguish $G$ and $H$.

**Baselines** On the molecular regression datasets, we compare the ET to an 1-WL expressive GNN baseline such as GIN(E) [51].

On ZINC (12K), ZINC-FULL and ALCHEMY, we compare the ET to other theoretically-aligned models, most notably higher-order GNNs [8, 36, 38], Graphormer-GD, with strictly less expressive power than the 3-WL [53], and PPGN++, with strictly more expressive power than the 3-WL [40] to study **Q1**. On PCQM4MV2, we compare the ET to state-of-the-art graph transformers to study **Q2**. To study the impact of positional/structural encodings in **Q3**, we evaluate the ET both with and without relative random walk probabilities (RRWP) positional encodings, recently proposed in Ma et al. [31]. RRWP encodings only apply to models with explicit representations over node pairs and are well-suited for the ET.

On the CLRS benchmark, we mostly compare to the Relational Transformer (RT) [12] as a strong graph transformer baseline. Comparing the ET to the RT allows us to study **Q2** in a different domain than molecular regression and on node- and edge-level tasks. Further, since the RT is similarly motivated as the ET in learning explicit representations of relations, we can study the potential benefits of the ET provable expressive power on the CLRS tasks. In addition, we compare the ET to DeepSet and GNN baselines in Diao and Loynd [12] and the single-task Triplet-GMPNN in Ibarz et al. [23].

On the BREC benchmark, we study questions **Q1** and **Q2** by comparing the ET to selected models presented in Wang and Zhang [48]. First, we compare to the $\delta$-2-LGNN [36], a higher-order GNN with strictly more expressive power than the 1-WL. Second, we compare to Graphormer [52], an empirically strong graph transformer. Third, we compare to PPGN [32] with the same expressive power as the ET. We additionally include the 3-WL results on the graphs in BREC to investigate how many 3-WL distinguishable graphs the ET can distinguish in BREC.

**Experimental setup** See Table 7 for an overview of the used hyperparameters.

For ZINC (12K), ZINC-FULL, and PCQM4MV2, we follow the hyperparameters in Ma et al. [31]. For ALCHEMY, we follow standard protocol and split the data according to Morris et al. [38]. Here, we simply adopt the hyper-parameters of ZINC (12K) from Ma et al. [31] but set the batch size to 64.

We choose the same hyper-parameters as the RT for the CLRS benchmark. Also, following the RT, we train for 10K steps and report results over 20 random seeds. To stay as close as possible to the experimental setup of our baselines, we integrate our Jax implementation of the ET as a processor into the latest version of the CLRS code base. In addition, we explore the OOD validation technique presented in Jung and Ahn [24], where we use larger graphs for the validation set to encourage size generalization. This technique can be used within the CLRS code base through the experiment parameters.

Finally, for BREC, we keep the default hyper-parameters and follow closely the setup used by Wang and Zhang [48] for PPGN. We found learning on BREC to be quite sensitive to architectural choices,

possibly due to the small dataset sizes. As a result, we use a linear layer for the FFN and additionally apply layer normalization onto $\boldsymbol{X}_{il}\boldsymbol{W}^{Q}$, $\boldsymbol{X}_{lj}\boldsymbol{W}^{K}$ in Equation (2) and $\boldsymbol{V}_{ilj}$ in Equation (3).

For ZINC (12K), ZINC-FULL, PCQM4MV2, CLRS, and BREC, we follow the standard train/validation/test splits. For ALCHEMY, we split the data according to the splits in Morris et al. [38], the same as our baselines.

All experiments were performed on a mix of A10, L40, and A100 NVIDIA GPUs. For each run, we used at most 8 CPU cores and 64 GB of RAM, with the exception of PCQM4MV2 and ZINC-FULL, which were trained on 4 L40 GPUs with 16 CPU cores and 256 GB RAM.

Table 5: Number of distinguished pairs of non-isomorphic graphs on the BREC benchmark over 10 random seeds with standard deviation. Baseline results (over 1 random seed) are taken from Wang and Zhang [48]. For reference, we also report the number of graphs distinguishable by 3-WL.

| **Model** | Basic | Regular | Extension | CFI | *All* |
|---|---|---|---|---|---|
| $\delta$-2-LGNN | 60 | 50 | 100 | 6 | **216** |
| PPGN | 60 | 50 | 100 | 23 | **233** |
| Graphormer | 16 | 12 | 41 | 10 | 79 |
| ET | 60 $\pm 0.0$ | 50 $\pm 0.0$ | 100 $\pm 0.0$ | 48.1 $\pm 1.9$ | **258.1** $\pm 1.9$ |
| 3-WL | 60 | 50 | 100 | 60 | 270 |

**Results and discussion** In the following, we answer questions **Q1** to **Q3**. We highlight **first**, **second**, and **third** best results in each table.

We compare results on the molecular regression datasets in Table 4. On ZINC (12K) and ALCHEMY, the ET outperforms all baselines, even without using positional/structural encodings, positively answering **Q1**. Interestingly, on ZINC-FULL, the ET, while still among the best models, does not show superior performance. Further, the RRWP encodings we employ on the graph-level datasets improve the performance of the ET on all three datasets, positively answering **Q3**. Moreover, in Table 2, we compare the ET with a variety of graph learning models on ZINC (12K), demonstrating that the ET is highly competitive with state-of-the-art models. We observe similarly positive results in Table 6 where the ET outperforms strong graph transformer baselines such as GRIT [31], GraphGPS [41] and Graphormer [52] on PCQM4MV2. As a result, we can positively answer **Q2**.

In Table 3, we compare results on CLRS where the ET performs best when averaging all tasks or when averaging all algorithm classes, improving over RT and Triplet-GMPNN. Additionally, the ET performs best on 4 algorithm classes and is among the top 3 in 7/8 algorithm classes. Interestingly, only some models are best on a majority of algorithm classes. These results indicate a benefit of the ETs' expressive power on this benchmark, adding to the answer of **Q2**. Further, see Table 8 in Appendix E.3 for additional results using the OOD validation technique.

Finally, on the BREC benchmark, we observe that the ET cannot distinguish all graphs distinguishable by 3-WL. At the same time, the ET distinguishes more graphs than PPGN, the other 3-WL expressive model, providing an additional positive answer to **Q1**; see Table 5. Moreover, the ET distinguishes

Table 6: Average validation MAE on the PCQM4MV2 benchmark over a single random seed.

| **Model** | Val. MAE (↓) | # Params |
|---|---|---|
| EGT [22] | 0.0869 | 89.3M |
| GraphGPS$_{\text{Small}}$ [41] | 0.0938 | 6.2M |
| GraphGPS$_{\text{Medium}}$ [41] | 0.0858 | 19.4M |
| TokenGT$_{\text{ORF}}$ [27] | 0.0962 | 48.6M |
| TokenGT$_{\text{Lap}}$ [27] | 0.0910 | 48.5M |
| Graphormer [52] | 0.0864 | 48.3M |
| GRIT [31] | 0.0859 | 16.6M |
| GPTrans-L | **0.0809** | 86.0M |
| ET | **0.0840** | 16.8M |
| ET$_{\text{+RRWP}}$ | **0.0832** | 16.8M |

Table 7: Hyperparameters of the Edge Transformer across all datasets.

| **Hyperparameter** | ZINC(12K) | ALCHEMY | ZINC-FULL | CLRS | BREC | PCQM4MV2 |
|---|---|---|---|---|---|---|
| Learning rate | 0.001 | 0.001 | 0.001 | 0.00025 | 0.0001 | 0.0002 |
| Grad. clip norm | 1.0 | 1.0 | 1.0 | 1.0 | – | 5.0 |
| Batch size | 32 | 64 | 256 | 4 | 16 | 256 |
| Optimizer | AdamW | Adam | AdamW | Adam | Adam | AdamW |
| Num. layers | 10 | 10 | 10 | 3 | 5 | 10 |
| Hidden dim. | 64 | 64 | 64 | 192 | 32 | 384 |
| Num. heads | 8 | 8 | 8 | 12 | 4 | 16 |
| Activation | GELU | GELU | GELU | RELU | – | GELU |
| Pooling | SUM | SUM | SUM | – | – | SUM |
| RRWP dim. | 32 | 32 | 32 | – | – | 128 |
| Weight decay | 1e-5 | 1e-5 | 1e-5 | – | 0.0001 | 0.1 |
| Dropout | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 |
| Attention dropout | 0.2 | 0.2 | 0.2 | 0.0 | 0.0 | 0.1 |
| # Steps | – | – | – | 10K | – | 2M |
| # Warm-up steps | – | – | – | 0 | – | 60K |
| # Epochs | 2K | 2K | 1K | – | 20 | – |
| # Warm-up epochs | 50 | 50 | 50 | – | 0 | – |
| # RRWP steps | 21 | 21 | 21 | – | – | 22 |

more graphs than $\delta$-2-LGNN and outperforms Graphormer by a large margin, again positively answering **Q2**. Overall, the positive results of the ET on BREC indicate that the ET is well able to leverage its expressive power empirically.

### E.1 Additional experimental details

Here, we give additional experimental details and results; see Table 7 for an overview of the selected hyper-parameters for all experiments.

See Appendix E.3 and Appendix E.4 for detailed results on the CLRS benchmark. Note that in the case of CLRS we evaluate in the single-task setting where we train a new set of parameters for each concrete algorithm, initially proposed in CLRS, to be able to fairly compare against graph transformers. We leave the multi-task learning proposed in Ibarz et al. [23] for future work.

### E.2 Data source and license

ZINC (12K), ALCHEMY (12K) and ZINC-FULL are available at `https://pyg.org` under an MIT license. PCQM4MV2 is available at `https://ogb.stanford.edu/docs/lsc/pcqm4mv2/` under a CC BY 4.0 license. The CLRS benchmark is available at `https://github.com/google-deepmind/clrs` under an Apache 2.0 license. The BREC benchmark is available at `https://github.com/GraphPKU/BREC` under an MIT license.

### E.3 Experimental results OOD validation in CLRS

In Table 8, following [24], we present additional experimental results on CLRS when using graphs of size 32 in the validation set. We compare to both the Triplet-GMPNN [23], as well as the TEAM [24] baselines. In addition, in Figure 3 we present a comparison on the improvements resulting from the OOD validation technique, comparing Triplet-GMPNN and the ET. Finally, in Table 9, we compare different modifications to the CLRS training setup that are agnostic to the choice of processor.

### E.4 CLRS test scores

Here, we present detailed results for the algorithms in CLRS; see Table 12 for divide and conquer algorithms, Table 13 for dynamic programming algorithms, Table 14 for geometry algorithms, Table 16 for greedy algorithms, Table 11 for search algorithms, Table 10 for sorting algorithms, Table 17 for string algorithms.

Table 8: Average test scores for the different algorithm classes and average test scores of all algorithms in CLRS **with the OOD validation technique** over 10 seeds; see Appendix E.4 for test scores per algorithm and Appendix E.5 for details on the standard deviation. Baseline results for Triplet-GMPNN and TEAM are taken from Jung and Ahn [24]. Results in %.

| **Algorithm** | Triplet-GMPNN | TEAM | ET (ours) |
|---|---|---|---|
| Sorting | 72.08 | 68.75 | **88.35** |
| Searching | 61.89 | 63.00 | **80.00** |
| DC | 65.70 | 69.79 | **74.70** |
| Greedy | 91.21 | **91.80** | 88.29 |
| DP | **90.08** | 83.61 | 84.69 |
| Graphs | 77.89 | 81.86 | **89.89** |
| Strings | 75.33 | **81.25** | 51.22 |
| Geometry | 88.02 | **94.03** | 89.68 |
| Avg. algorithm class | 77.48 | 79.23 | **80.91** |
| All algorithms | 78.00 | 79.82 | **85.01** |

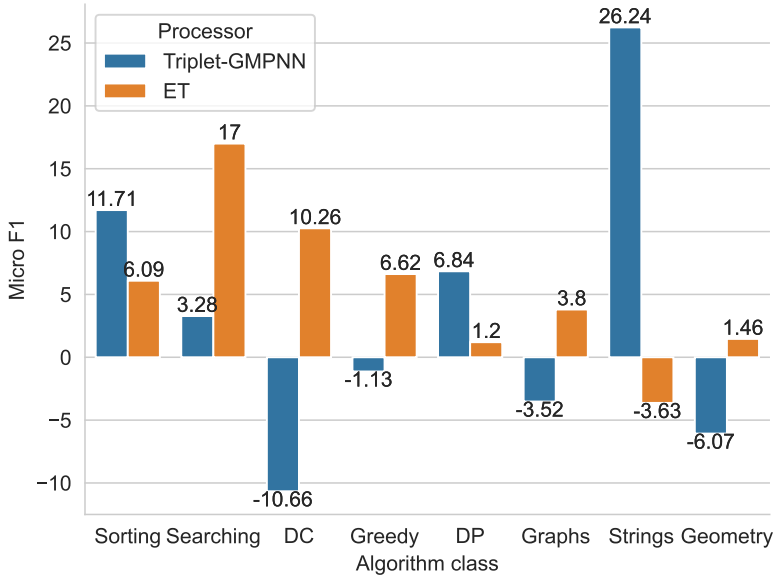



Figure 3: Difference in micro F1 with and without the OOD validation technique in Jung and Ahn [24], for Triplet-GMPNN [23] and ET, respectively.

Table 9: CLRS-30 Processor-agnostic modifications

| **Processor** | Markov [9] | OOD Validation [24] | *Avg. algorithm class* | *All algorithms* |
|---|---|---|---|---|
| Triplet-GMPNN | ✓ | ✗ | 79.75 | 82.89 |
| Triplet-GMPNN | ✗ | ✓ | 77.65 | 78.00 |
| TEAM | ✗ | ✓ | 79.23 | 79.82 |
| ET | ✗ | ✓ | **80.91** | **85.02** |

Table 10: Detailed test scores for the ET on sorting algorithms.

| **Algorithm** | F1-score(%) | Std. dev.(%) | F1-score(%)(OOD) | Std. dev.(%) (OOD) |
|---|---|---|---|---|
| Bubble Sort | 93.60 | 3.87 | 87.44 | 13.48 |
| Heapsort | 64.36 | 22.41 | 80.96 | 12.97 |
| Insertion Sort | 85.71 | 20.68 | 91.74 | 6.83 |
| Quicksort | 85.37 | 8.70 | 93.25 | 9.10 |
| *Average* | 82.26 | 13.92 | 88.35 | 10.58 |

Table 11: Detailed test scores for the ET on search algorithms.

| **Algorithm** | F1-score(%) | Std. dev.(%) | F1-score(%)(OOD) | Std. dev.(%) (OOD) |
|---|---|---|---|---|
| Binary Search | 79.96 | 11.66 | 90.84 | 2.71 |
| Minimum | 96.88 | 1.74 | 97.94 | 0.87 |
| Quickselect | 12.43 | 11.72 | 52.64 | 22.04 |
| *Average* | 63.00 | 8.00 | 80.00 | 8.54 |

Table 12: Detailed test scores for the ET on divide and conquer algorithms.

| **Algorithm** | F1-score(%) | Std. dev.(%) | F1-score(%)(OOD) | Std. dev.(%) (OOD) |
|---|---|---|---|---|
| Find Max. Subarray Kadande | 64.44 | 2.24 | 74.70 | 2.59 |
| *Average* | 64.44 | 2.24 | 74.70 | 2.59 |

Table 13: Detailed test scores for the ET on dynamic programming algorithms.

| **Algorithm** | F1-score(%) | Std. dev.(%) | F1-score(%)(OOD) | Std. dev.(%) (OOD) |
|---|---|---|---|---|
| LCS Length | 88.67 | 2.05 | 88.97 | 2.06 |
| Matrix Chain Order | 90.11 | 3.28 | 90.84 | 2.94 |
| Optimal BST | 71.70 | 5.46 | 74.26 | 10.84 |
| *Average* | 83.49 | 3.60 | 84.68 | 5.28 |

Table 14: Detailed test scores for the ET on geometry algorithms.

| **Algorithm** | F1-score(%) | Std. dev.(%) | F1-score(%)(OOD) | Std. dev.(%) (OOD) |
|---|---|---|---|---|
| Graham Scan | 92.23 | 2.26 | 96.09 | 0.96 |
| Jarvis March | 89.09 | 8.92 | 95.18 | 1.46 |
| Segments Intersect | 83.35 | 7.01 | 77.78 | 1.16 |
| *Average* | 88.22 | 6.09 | 89.68 | 1.19 |

Table 15: Detailed test scores for the ET on graph algorithms.

| **Algorithm** | F1-score(%) | Std. dev.(%) | F1-score(%)(OOD) | Std. dev.(%) (OOD) |
|---|---|---|---|---|
| Articulation Points | 93.06 | 0.62 | 95.47 | 2.35 |
| Bellman-Ford | 89.96 | 3.77 | 95.55 | 1.65 |
| BFS | 99.77 | 0.30 | 99.95 | 0.08 |
| Bridges | 91.95 | 10.00 | 98.28 | 2.64 |
| DAG Shortest Paths | 97.63 | 0.85 | 98,43 | 0.65 |
| DFS | 65.60 | 17.98 | 57.76 | 14.54 |
| Dijkstra | 91.90 | 2.99 | 97.32 | 7.32 |
| Floyd-Warshall | 61.53 | 5.34 | 83.57 | 1.79 |
| MST-Kruskal | 84.06 | 2.14 | 87.21 | 1.45 |
| MST-Prim | 93.02 | 2.41 | 93.00 | 1.61 |
| SCC | 65.80 | 8.13 | 74.58 | 5.31 |
| Topological Sort | 98.74 | 2.24 | 97.53 | 2.31 |
| *Average* | 86.08 | 4.73 | 89.92 | 3.02 |

Table 16: Detailed test scores for the ET on greedy algorithms.

| **Algorithm** | F1-score(%) | Std. dev.(%) | F1-score(%)(OOD) | Std. dev.(%) (OOD) |
|---|---|---|---|---|
| Activity Selector | 80.12 | 12.34 | 91.72 | 2.35 |
| Task Scheduling | 83.21 | 0.30 | 84.85 | 2.83 |
| *Average* | 81.67 | 6.34 | 88.28 | 2.59 |

### E.5 CLRS test standard deviation

Here, we compare the standard deviation of Deep Sets, GAT, MPNN, PGN, RT, and ET following the comparison in Diao and Loynd [12]; see Table 18. We observe that the ET has the lowest overall standard deviation. Note that we omit Triplet-GMPNN [23] since we do not have access to the test results for each algorithm on each seed that are necessary to compute the overall standard deviation. Instead, we compare the standard deviation per algorithm class between Triplet-GMPNN and the ET in Table 19. We observe that Triplet-GMPNN and the ET have comparable standard deviations except for search and string algorithms, where Triplet-GMPNN has a much higher standard deviation than the ET.

## F Runtime and memory

Here, we provide additional information on the runtime and memory requirements of the ET in practice. Specifically, in Figure 4, we provide runtime scaling of the ET with and without low-level GPU optimizations in PyTorch on an A100 GPU with `bfloat16` precision. We measure the time for the forward pass of a single layer of the ET on a single graph (batch size of 1) with $n \in \{100, 200, ..., 700\}$ nodes and average the runtime over 100 repeats. We sample random Erdős-Renyi graphs with edge probability $0.05$. We use an embedding dimension of 64 and two attention heads. We find that the automatic compilation into Triton [44], performed by automatically by `torch.compile`, improves the runtime and memory scaling. Specifically, with `torch.compile` enabled, the ET layer can process graphs with up to 700 nodes and shows much more efficient runtime scaling with the number of nodes.

Table 17: Detailed test scores for the ET on string algorithms.

| **Algorithm** | F1-score(%) | Std. dev.(%) | F1-score(%)(OOD) | Std. dev.(%) (OOD) |
|---|---|---|---|---|
| KMP Matcher | 10.47 | 10.28 | 8.67 | 8.14 |
| Naive String Match | 99.21 | 1.10 | 93.76 | 6.28 |
| *Average* | 54.84 | 5.69 | 51.21 | 7.21 |

Table 18: Standard deviation of Deep Sets, GAT, MPNN, PGN, RT and ET (over all algorithms and all seeds).

| **Model** | Std. Dev. (%) |
|---|---|
| Deep Sets | 29.3 |
| GAT | 32.3 |
| MPNN | 34.6 |
| PGN | 33.1 |
| RT | 29.6 |
| ET | **26.6** |

Table 19: Standard deviation per algorithm class of Triplet-GMPNN (over 10 random seeds) as reported in Ibarz et al. [23] and ET (over 10 random seeds). Results in %.

| **Algorithm class** | Triplet-GMPNN | ET |
|---|---|---|
| Sorting | 12.16 | 15.57 |
| Searching | 24.34 | 3.51 |
| Divide and Conquer | 1.34 | 2.46 |
| Greedy | 2.95 | 6.54 |
| Dynamic Programming | 4.98 | 3.60 |
| Graphs | 6.21 | 6.79 |
| Strings | 23.49 | 8.60 |
| Geometry | 2.30 | 3.77 |
| *Average* | 9.72 | **6.35** |

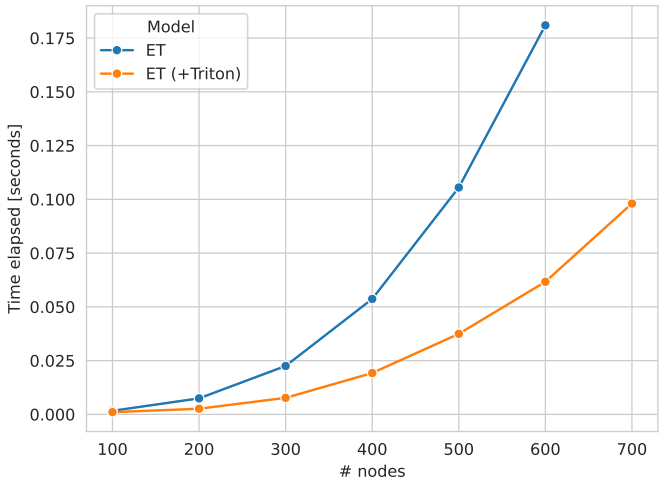



Figure 4: Runtime of the forward pass of a single ET layer in PyTorch in seconds for graphs with up to 700 nodes. We compare the runtime with and without `torch.compile` (automatic compilation into Triton [44]) enabled. Without compilation, the ET goes out of memory after 600 nodes.

Table 20: Runtime of a single run of the ET in CLRS on a single A100 GPU.

| Algorithm | Time in hh:mm:ss |
|---|---|
| Activity Selector | 00:09:38 |
| Articulation Points | 01:19:39 |
| Bellman Ford | 00:07:55 |
| BFS | 00:07:03 |
| Binary Search | 00:05:53 |
| Bridges | 01:20:44 |
| Bubble Sort | 01:05:34 |
| DAG Shortest Paths | 00:29:15 |
| DFS | 00:27:47 |
| Dijkstra | 00:09:37 |
| Find Maximum Subarray Kadane | 00:15:25 |
| Floyd Warshall | 00:12:56 |
| Graham Scan | 00:15:55 |
| Heapsort | 00:57:14 |
| Insertion Sort | 00:10:39 |
| Jarvis March | 01:34:40 |
| Kmp Matcher | 00:57:56 |
| LCS Length | 00:08:12 |
| Matrix Chain Order | 00:15:31 |
| Minimum | 00:21:25 |
| MST Kruskal | 01:15:54 |
| MST Prim | 00:09:34 |
| Naive String Matcher | 00:51:05 |
| Optimal BST | 00:12:57 |
| Quickselect | 02:25:03 |
| Quicksort | 00:59:24 |
| Segments Intersect | 00:03:38 |
| Strongly Connected Components | 00:56:58 |
| Task Scheduling | 00:08:50 |
| Topological Sort | 00:27:40 |

Table 21: Runtime of a single run on the molecular regression datasets, as well as BREC, on L40 GPUs in *days:hours:minutes:seconds*.

| | ZINC (12K) | ALCHEMY (12K) | ZINC-FULL | PCQM4MV2 | BREC |
|---|---|---|---|---|---|
| ET | 00:06:04:52 | 00:02:47:51 | 00:23:11:05 | 03:10:35:11 | 00:00:08:37 |
| ET+RRWP | 00:06:19:52 | 00:02:51:23 | 01:01:10:55 | 03:10:22:06 | - |
| Num. GPUs | 1 | 1 | 4 | 4 | 1 |

**Hardware optimizations** Efficient compilation of neural networks is already available via compilers such as Triton [44]. In addition, we want to highlight `FlashAttention` [11] available for the standard transformer as an example of architecture-specific hardware optimizations that can reduce runtime and memory requirements.

**Runtime per dataset/benchmark** Here, we present additional runtime results for all of our datasets. We present the runtime of a single run on a single L40 GPU of ZINC (12K), ALCHEMY (12K), and BREC. For ZINC-FULL and PCQM4MV2, we present the runtime of a single run on 4 L40 GPUs; see Table 21.

On CLRS, the experiments in our work are run on a mix of A10 and A100 GPUs. To enable a fair comparison, we rerun each algorithm in CLRS in a single run on a single A100 GPU and report the corresponding runtime in Table 20. Finally, we note that these numbers only reflect the time to run the final experiments and significantly more time was used for preliminary experiments over the course of the research project.

# G Limitations

While proving to be a strong and versatile graph model, the ET has an asymptotic runtime and memory complexity of $\mathcal{O}(n^3)$ which is more expensive than most state-of-the-art models with linear or quadratic runtime and memory complexity. We emphasize that due to the runtime and memory complexity of the $k$-$\mathsf{WL}$, a trade-off between expressivity and efficiency is likely unavoidable. At the same time, the ET is highly parallelizable and runs efficiently on modern GPUs. We hope that innovations for parallelizable neural networks can compensate for the asymptotic runtime and memory complexity of the ET. In Figure 4 in the appendix, we find that we can use low-level GPU optimizations, available for parallelizable neural networks out-of-the-box, to dampen the cubic runtime and memory scaling of the ET; see Appendix F for runtime and memory experiments and an extended discussion.

# H Extended preliminaries

Here, we define our notation. Let $\mathbb{N} \coloneqq \{1, 2, 3, \dots\}$. For $n \geq 1$, let $[n] \coloneqq \{1, \dots, n\} \subset \mathbb{N}$. We use $\{\!\{ \dots \}\!\}$ to denote multisets, i.e., the generalization of sets allowing for multiple instances for each of its elements.

**Graphs** A *(node-)labeled graph* $G$ is a triple $(V(G), E(G), \ell)$ with *finite* sets of *vertices* or *nodes* $V(G)$, *edges* $E(G) \subseteq \{\{u, v\} \subseteq V(G) \mid u \neq v\}$ and a (node-)label function $\ell \colon V(G) \to \mathbb{N}$. Then $\ell(v)$ is a *label* of $v$, for $v$ in $V(G)$. If not otherwise stated, we set $n \coloneqq |V(G)|$, and the graph is of *order* $n$. We also call the graph $G$ an $n$-order graph. For ease of notation, we denote the edge $\{u, v\}$ in $E(G)$ by $(u, v)$ or $(v, u)$. We define an $n$-order *attributed graph* as a pair $\mathcal{G} = (G, \boldsymbol{F})$, where $G = (V(G), E(G))$ and $\boldsymbol{F}$ in $\mathbb{R}^{n \times p}$ for $p > 0$ is a *node feature matrix*. Here, we identify $V(G)$ with $[n]$, then $\boldsymbol{F}(v)$ in $\mathbb{R}^{1 \times p}$ is the *feature* or *attribute* of the node $v \in V(G)$. Given a labeled graph $(V(G), E(G), \ell)$, a node feature matrix $\boldsymbol{F}$ is *consistent* with $\ell$ if $\ell(v) = \ell(w)$ for $v, w \in V(G)$ if, and only, if $\boldsymbol{F}(v) = \boldsymbol{F}(w)$.

**Neighborhood and Isomorphism** The *neighborhood* of a vertex $v$ in $V(G)$ is denoted by $N(v) \coloneqq \{u \in V(G) \mid (v, u) \in E(G)\}$ and the *degree* of a vertex $v$ is $|N(v)|$. Two graphs $G$ and $H$ are *isomorphic* and we write $G \simeq H$ if there exists a bijection $\varphi \colon V(G) \to V(H)$ preserving the adjacency relation, i.e., $(u, v)$ is in $E(G)$ if, and only, if $(\varphi(u), \varphi(v))$ is in $E(H)$. Then $\varphi$ is an *isomorphism* between $G$ and $H$. In the case of labeled graphs, we additionally require that $l(v) = l(\varphi(v))$ for $v$ in $V(G)$, and similarly for attributed graphs. Moreover, we call the equivalence classes induced by $\simeq$ *isomorphism types* and denote the isomorphism type of $G$ by $\tau_G$. We further define the atomic type $\mathrm{atp} \colon V(G)^k \to \mathbb{N}$, for $k > 0$, such that $\mathrm{atp}(\boldsymbol{v}) = \mathrm{atp}(\boldsymbol{w})$ for $\boldsymbol{v}$ and $\boldsymbol{w}$ in $V(G)^k$ if, and only, if the mapping $\varphi \colon V(G)^k \to V(G)^k$ where $v_i \mapsto w_i$ induces a partial isomorphism, i.e., we have $v_i = v_j \iff w_i = w_j$ and $(v_i, v_j) \in E(G) \iff (\varphi(v_i), \varphi(v_j)) \in E(G)$.

**Matrices** Let $\boldsymbol{M} \in \mathbb{R}^{n \times p}$ and $\boldsymbol{N} \in \mathbb{R}^{n \times q}$ be two matrices then $[\boldsymbol{M} \quad \boldsymbol{N}] \in \mathbb{R}^{n \times p+q}$ denotes column-wise matrix concatenation. We also write $\mathbb{R}^d$ for $\mathbb{R}^{1 \times d}$. Further, let $\boldsymbol{M} \in \mathbb{R}^{p \times n}$ and

$\boldsymbol{N} \in \mathbb{R}^{q \times n}$ be two matrices then

$$\begin{bmatrix} \boldsymbol{M} \\ \boldsymbol{N} \end{bmatrix} \in \mathbb{R}^{p+q \times n}$$

denotes row-wise matrix concatenation.

For a matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, we denote with $\boldsymbol{X}_i$ the $i$th row vector. In the case where the rows of $\boldsymbol{X}$ correspond to nodes in a graph $G$, we use $\boldsymbol{X}_v$ to denote the row vector corresponding to the node $v \in V(G)$.

**The Weisfeiler–Leman algorithm** We describe the Weisfeiler–Leman algorithm, starting with the 1-$\mathsf{WL}$. The 1-$\mathsf{WL}$ or color refinement is a well-studied heuristic for the graph isomorphism problem, originally proposed by Weisfeiler and Leman [49].[1] Intuitively, the algorithm determines if two graphs are non-isomorphic by iteratively coloring or labeling vertices. Formally, let $G = (V, E, \ell)$ be a labeled graph, in each iteration, $t > 0$, the 1-$\mathsf{WL}$ computes a vertex coloring $C^1_t \colon V(G) \to \mathbb{N}$, depending on the coloring of the neighbors. That is, in iteration $t > 0$, we set

$$C^1_t(v) \coloneqq \mathsf{recolor}\Big(\big(C^1_{t-1}(v), \{\!\{C^1_{t-1}(u) \mid u \in N(v)\}\!\}\big)\Big),$$

for all vertices $v$ in $V(G)$, where $\mathsf{recolor}$ injectively maps the above pair to a unique natural number, which has not been used in previous iterations. In iteration 0, the coloring $C^1_0 \coloneqq \ell$. To test if two graphs $G$ and $H$ are non-isomorphic, we run the above algorithm in "parallel" on both graphs. If the two graphs have a different number of vertices colored $c$ in $\mathbb{N}$ at some iteration, the 1-$\mathsf{WL}$ *distinguishes* the graphs as non-isomorphic. It is easy to see that the algorithm cannot distinguish all non-isomorphic graphs [10]. Several researchers, e.g., Babai [3], Cai et al. [10], devised a more powerful generalization of the former, today known as the $k$-dimensional Weisfeiler–Leman algorithm ($k$-$\mathsf{WL}$), operating on $k$-tuples of vertices rather than single vertices.

**The $k$-dimensional Weisfeiler–Leman algorithm** Due to the shortcomings of the 1-$\mathsf{WL}$ or color refinement in distinguishing non-isomorphic graphs, several researchers, e.g., Babai [3], Cai et al. [10], devised a more powerful generalization of the former, today known as the $k$-dimensional Weisfeiler-Leman algorithm ($k$-$\mathsf{WL}$), operating on $k$-tuples of nodes rather than single nodes.

Intuitively, to surpass the limitations of the 1-$\mathsf{WL}$, the $k$-$\mathsf{WL}$ colors node-ordered $k$-tuples instead of a single node. More precisely, given a graph $G$, the $k$-$\mathsf{WL}$ colors the tuples from $V(G)^k$ for $k \geq 2$ instead of the nodes. By defining a neighborhood between these tuples, we can define a coloring similar to the 1-$\mathsf{WL}$. Formally, let $G$ be a graph, and let $k \geq 2$. In each iteration, $t \geq 0$, the algorithm, similarly to the 1-$\mathsf{WL}$, computes a *coloring* $C^k_t \colon V(G)^k \to \mathbb{N}$. In the first iteration, $t = 0$, the tuples $\boldsymbol{v}$ and $\boldsymbol{w}$ in $V(G)^k$ get the same color if they have the same atomic type, i.e., $C^k_0(\boldsymbol{v}) \coloneqq \mathsf{atp}(\boldsymbol{v})$. Then, for each iteration, $t > 0$, $C^k_t$ is defined by

$$C^k_t(\boldsymbol{v}) \coloneqq \mathsf{recolor}\big(C^k_{t-1}(\boldsymbol{v}), M_t(\boldsymbol{v})\big), \tag{5}$$

with $M_t(\boldsymbol{v})$ the multiset

$$M_t(\boldsymbol{v}) \coloneqq \big(\{\!\{C^k_{t-1}(\phi_1(\boldsymbol{v}, w)) \mid w \in V(G)\}\!\}, \ldots, \{\!\{C^k_{t-1}(\phi_k(\boldsymbol{v}, w)) \mid w \in V(G)\}\!\}\big), \tag{6}$$

and where

$$\phi_j(\boldsymbol{v}, w) \coloneqq (v_1, \ldots, v_{j-1}, w, v_{j+1}, \ldots, v_k).$$

That is, $\phi_j(\boldsymbol{v}, w)$ replaces the $j$-th component of the tuple $\boldsymbol{v}$ with the node $w$. Hence, two tuples are *adjacent* or *$j$-neighbors* if they are different in the $j$th component (or equal, in the case of self-loops). Hence, two tuples $\boldsymbol{v}$ and $\boldsymbol{w}$ with the same color in iteration $(t-1)$ get different colors in iteration $t$ if there exists a $j$ in $[k]$ such that the number of $j$-neighbors of $\boldsymbol{v}$ and $\boldsymbol{w}$, respectively, colored with a certain color is different.

We run the $k$-$\mathsf{WL}$ algorithm until convergence, i.e., until for $t$ in $\mathbb{N}$

$$C^k_t(\boldsymbol{v}) = C^k_t(\boldsymbol{w}) \iff C^k_{t+1}(\boldsymbol{v}) = C^k_{t+1}(\boldsymbol{w}),$$

[1] Strictly speaking, the 1-$\mathsf{WL}$ and color refinement are two different algorithms. The 1-$\mathsf{WL}$ considers neighbors and non-neighbors to update the coloring, resulting in a slightly higher expressive power when distinguishing vertices in a given graph; see [18] for details. For brevity, we consider both algorithms to be equivalent.

for all $\boldsymbol{v}$ and $\boldsymbol{w}$ in $V(G)^k$ holds.

Similarly to the 1-WL, to test whether two graphs $G$ and $H$ are non-isomorphic, we run the $k$-WL in "parallel" on both graphs. Then, if the two graphs have a different number of nodes colored $c$, for $c$ in $\mathbb{N}$, the $k$-WL *distinguishes* the graphs as non-isomorphic. By increasing $k$, the algorithm gets more powerful in distinguishing non-isomorphic graphs, i.e., for each $k \geq 2$, there are non-isomorphic graphs distinguished by $(k+1)$-WL but not by $k$-WL [10]. We now also define the folklore $k$-WL hierarchy.

**The folklore $k$-dimensional Weisfeiler–Leman algorithm** A common and well-studied variant of the $k$-WL is the $k$-FWL, which differs from the $k$-WL only in the aggregation function. Instead of Equation (6), the "folklore" version of the $k$-WL updates $k$-tuples according to

$$M_t^{\mathrm{F}}(\boldsymbol{v}) \coloneqq \{\!\{(C_{t-1}^{k,\mathrm{F}}(\phi_1(\boldsymbol{v}, w)), ..., C_{t-1}^{k,\mathrm{F}}(\phi_k(\boldsymbol{v}, w))) \mid w \in V(G)\}\!\},$$

resulting in the coloring $C_t^{k,\mathrm{F}} \colon V(G)^k \to \mathbb{N}$, and is strictly more powerful than the $k$-WL. Specifically, for $k \geq 2$, the $k$-WL is exactly as powerful as the $(k-1)$-FWL [18].

**Computing $k$-WL's initial colors** Let $G = (V(G), E(G))$ be a graph, $k \geq 2$, and $\boldsymbol{u} \in V(G)^k$ be a $k$-tuple. Then we can present the atomic type $\mathrm{atp}(\boldsymbol{v})$ by a $k \times k$ matrix $K$ over $\{1, 2, 3\}$. That is, the entry $K_{ij}$ is 1 if $(v_i, v_j) \in E(G)$, 2 if $v_i = v_j$, and 3 otherwise.

### H.1 Relationship between first-order logic and Weisfeiler–Leman

We begin with a short review of Cai et al. [10]. We consider our usual node-labeled graph $G = (V(G), E(G), \ell)$ with $n$ nodes. However, we replace $\ell$ with a countable set of color relations $C_1, \ldots, C_n$, where for a node $v \in V(G)$,

$$C_i(v) \iff \ell(v) = i.$$

Note that Cai et al. [10] consider the more general case where nodes can be assigned to multiple colors simultaneously. However, for our work, we assume that a node is assigned to precisely one color, and hence, the set of color relations is at most of size $n$. We can construct first-order logic statements about $G$. For example, the following sentence describes a graph with a triangle formed by two nodes with color 1:

$$\exists x_1 \exists x_2 \exists x_3 \big(E(x_1, x_2) \wedge E(x_1, x_3) \wedge E(x_2, x_3) \wedge C_1(x_1) \wedge C_1(x_2)\big).$$

Here, $x_1$, $x_2$, and $x_3$ are *variables* which can be repeated and re-quantified at will. Statements made about $G$ and a subset of nodes in $V(G)$ are of particular importance to us. To this end, we define a *$k$-configuration*, a partial function $f : \{x_1, \ldots, x_k\} \to V(G)$ that assigns a node in $V(G)$ to one of $k$ variables. Let $\varphi$ be a first-order sentence with free variables $x_1, \ldots, x_l$, where $l \leq k$. Then, we write

$$G, f \models \varphi$$

if $\varphi$ is true for nodes $f(x_1), \ldots, f(x_l)$. Cai et al. [10] now define the language $\mathcal{C}_k$ of all first-order sentences with at most $k$ variables and counting quantifiers. For example, the following sentence in $\mathcal{C}_k$ lets us describe a graph with exactly 3 triangles where one node has color 3:

$$\exists !3 x_1 \exists x_2 \exists x_3 \big(E(x_1, x_2) \wedge E(x_1, x_3) \wedge E(x_2, x_3) \wedge C_3(x_1)\big).$$

Given two graphs $G$ and $H$ and respective $k$-configurations $f$ and $g$ with the same domain, we can now define an equivalence class $\equiv_{\mathcal{C}_k}$ and say that $G, f$ and $H, g$ are $\mathcal{C}_k$ equivalent, denoted

$$G, f \equiv_{\mathcal{C}_k} H, g$$

if and only if for all $\varphi \in \mathcal{C}_k$ with at most $k$ free variables,

$$G, f \models \varphi \iff H, g \models \varphi.$$

It is important to note that configurations are used to define a concept beyond the scope of a concrete graph. Instead, a concept can be applied to a graph $G$ by defining a configuration, i.e., a mapping between concrete nodes in $G$ to abstract logical variables.

Now, Cai et al. [10] prove that, given two graphs $G$ and $H$ and respective $k$-configurations $f$ and $g$, we have that $\boldsymbol{u} \coloneqq (f(x_1), \ldots, f(x_k)) \in V(G)^k$ and $\boldsymbol{v} \coloneqq (g(x_1), \ldots, g(x_k)) \in V(G)^k$ are $k$-tuples and

$$C_t^{k,\mathrm{F}}(\boldsymbol{u}) = C_t^{k,\mathrm{F}}(\boldsymbol{v}) \iff G, f \equiv_{\mathcal{C}_{k+1}} H, g,$$

for some $t \geq 0$. Using Theorem 1, we can then also say that there exists a parameterization of the ET such that

$$\boldsymbol{X}^{(t)}(\boldsymbol{u}) = \boldsymbol{X}^{(t)}(\boldsymbol{v}) \iff G, f \equiv_{\mathcal{C}_3} H, g.$$

# I Intuition on our results

In the following, we provide some intuition of how the ET can simulate the 2-FWL. Given a tuple $(i, j) \in V(G)^2$, we encode its color at iteration $t$ with $\boldsymbol{X}_{ij}^{(t)}$. Further, to represent the multiset

$$\{\!\{(C_{t-1}^{2,\mathrm{F}}(i,l),\, C_{t-1}^{2,\mathrm{F}}(l,j)) \mid l \in V(G)\}\!\},$$

we show that it is possible to encode the pair of colors

$$(C_{t-1}^{2,\mathrm{F}}(i,l),\, C_{t-1}^{2,\mathrm{F}}(l,j)) \quad \text{via} \quad \boldsymbol{X}_{il}^{(t-1)}\boldsymbol{W}^{V_1} \odot \boldsymbol{X}_{lj}^{(t-1)}\boldsymbol{W}^{V_2},$$

for node $l \in V(G)$. Finally, triangular attention in Equation (1), performs weighted sum aggregation over the 2-tuple of colors $(C_{t-1}^{2,\mathrm{F}}(i,l),\, C_{t-1}^{2,\mathrm{F}}(l,j))$ for each $l$, which we show is sufficient to represent the multiset; see Appendix J. For the other direction, namely that the ET has at most 3-WL expressive power, we simply show that the recolor function can simulate the value fusion in Equation (3), as well as the triangular attention in Equation (1).

# J Proofs

Here, we first generalize the GNN from Grohe [18] to the 2-FWL. Higher-order GNNs with the same expressivity have been proposed in prior works by Azizian and Lelarge [1]. However, our GNNs have a special form that can be computed by the Edge Transformer.

Formally, let $S \subseteq \mathbb{N}$ be a finite subset. First, we show that multisets over $S$ can be injectively mapped to a value in the closed interval $(0, 1)$, a variant of Lemma VIII.5 in Grohe [18]. Here, we outline a streamlined version of its proof, highlighting the key intuition behind representing multisets as $m$-ary numbers. Let $M \subseteq S$ be a multiset with multiplicities $a_1, \ldots, a_k$ and distinct $k$ values. We define the *order* of the multiset as $\sum_{i=1}^{k} a_i$. We can write such a multiset as a sequence $x^{(1)}, \ldots, x^{(l)}$ where $l$ is the order of the multiset. Note that the order of the sequence is arbitrary and that for $i \neq j$ it is possible to have $x^{(i)} = x^{(j)}$. We call such a sequence an $M$-sequence of length $l$. We now prove a slight variation of a result of Grohe [18].

**Lemma 4.** *For a finite $m \in \mathbb{N}$, let $M \subseteq S$ be a multiset of order $m-1$ and let $x_i \in S$ denote the $i$th number in a fixed but arbitrary ordering of $S$. Given a mapping $g\colon S \to (0,1)$ where*

$$g(x_i) \coloneqq m^{-i},$$

*and an $M$-sequence of length $l$ given by $x^{(1)}, \ldots, x^{(l)}$ with positions $i^{(1)}, \ldots, i^{(l)}$ in $S$, the sum*

$$\sum_{j \in [l]} g(x^{(j)}) = \sum_{j \in [l]} m^{-i^{(j)}}$$

*is unique for every unique $M$.*

*Proof.* By assumption, let $M \subseteq S$ denote a multiset of order $m-1$. Further, let $x^{(1)}, \ldots, x^{(l)} \in M$ be an $M$-sequence with $i^{(1)}, \ldots, i^{(l)}$ in $S$. Given our fixed ordering of the numbers in $S$ we can equivalently write $M = ((a_1, x_1), \ldots, (a_n, x_n))$, where $a_i$ denotes the multiplicity of $i$th number in $M$ with position $i$ from our ordering over $S$. Note that for a number $m^{-i}$ there exists a corresponding $m$-ary number written as

$$0.0 \ldots \underbrace{1}_{i} \ldots$$

Then the sum,

$$\begin{aligned}\sum_{j \in [l]} g(x^{(j)}) &= \sum_{j \in [l]} m^{-i^{(j)}} \\ &= \sum_{i \in S} a_i m^{-i} \in (0,1)\end{aligned}$$

and in $m$-ary representation

$$0.a_1 \ldots a_n.$$

Note that $a_i = 0$ if and only if there exists no $j$ such that $i^{(j)} = i$. Since the order of $M$ is $m - 1$, it holds that $a_i < m$. Hence, it follows that the above sum is unique for each unique multiset $M$, implying the result. □

Recall that $S \subseteq \mathbb{N}$ and that we fixed an arbitrary ordering over $S$. Intuitively, we use the finiteness of $S$ to map each number therein to a fixed digit of the numbers in $(0, 1)$. The finite $m$ ensures that at each digit, we have sufficient "bandwidth" to encode each $a_i$. Now that we have seen how to encode multisets over $S$ as numbers in $(0, 1)$, we review some fundamental operations about the $m$-ary numbers defined above. We will refer to decimal numbers $m^{-i}$ as *corresponding* to an $m$-ary number

$$0.0 \ldots \underbrace{1}_{i} \ldots,$$

where the $i$th digit after the decimal point is $1$ and all other digits are $0$, and vice versa.

To begin with, addition between decimal numbers implements *counting* in $m$-ary notation, i.e.,

$$m^{-i} + m^{-j} \text{ corresponds to } 0.0 \ldots \underbrace{1}_{i} \ldots \underbrace{1}_{j} \ldots,$$

for digit positions $i \neq j$ and

$$m^{-i} + m^{-j} \text{ corresponds to } 0.0 \ldots \underbrace{2}_{i=j} \ldots,$$

otherwise. We used counting in the previous result's proof to represent a multiset's multiplicities. Next, multiplication between decimal numbers implements *shifting* in $m$-ary notation, i.e.,

$$m^{-i} \cdot m^{-j} \text{ corresponds to } 0.0 \ldots \underbrace{1}_{i+j} \ldots.$$

Shifting further applies to general decimal numbers in $(0, 1)$. Let $x \in (0, 1)$ correspond to an $m$-ary number with $l$ digits,

$$0.a_1 \ldots a_l.$$

Then,

$$m^{-i} \cdot x \text{ corresponds to } 0.0 \ldots 0 \underbrace{a_1 \ldots a_l}_{i+1,\ldots,i+l}.$$

Before we continue, we show a small lemma stating that two non-overlapping sets of $m$-ary numbers preserve their uniqueness under addition.

**Lemma 5.** *Let $A$ and $B$ be two sets of $m$-ary numbers for some $m > 1$. If*

$$\min_{x \in A} x > \max_{y \in B} y,$$

*then for any $x_1, x_2 \in A, y_1, y_2 \in B$,*

$$x_1 + y_1 = x_2 + y_2 \iff x_1 = x_2 \text{ and } y_1 = y_2.$$

*Proof.* The statement follows from the fact that if

$$\min_{x \in A} x > \max_{y \in B} y,$$

then numbers in $A$ and numbers in $B$ do not overlap in terms of their digit range. Specifically, there exists some $l > 0$ such that we can write

$$x \coloneqq 0.x_1 \ldots x_l$$
$$y \coloneqq 0.\underbrace{0 \ldots 0}_{l} y_1 \ldots y_k,$$

for some $k > l$ and all $x \in A$, $y \in B$. As a result,

$$x + y = 0.x_1 \ldots x_l y_1 \ldots y_k.$$

Hence, $x + y$ is unique for every unique pair $(x, y)$. This completes the proof. □

We begin by showing the following proposition, showing that the tokenization in Equation (4) is sufficient to encode the initial node colors under 2-FWL.

**Proposition 6.** *Let $G = (V(G), E(G), \ell)$ be a node-labeled graph with $n$ nodes. Then, there exists a parameterization of Equation* (4) *with $d = 1$ such that for each 2-tuples $\boldsymbol{u}, \boldsymbol{v} \in V(G)^2$,*

$$C_0^{2,F}(\boldsymbol{u}) = C_0^{2,F}(\boldsymbol{v}) \iff \boldsymbol{X}(\boldsymbol{u}) = \boldsymbol{X}(\boldsymbol{v}).$$

*Proof.* The statement directly follows from the fact that the initial color of a tuple $\boldsymbol{u} \coloneqq (i, j)$ depends on the atomic type and the node labeling. In Equation (4), we encode the atomic type with $\boldsymbol{E}_{ij}$ and the node labels with

$$[\boldsymbol{E}_{ij} \quad \boldsymbol{F}_i \quad \boldsymbol{F}_j]$$

The concatenation of both node labels and atomic type is clearly injective. Finally, since there are at most $n^2$ distinct initial colors of the 2-FWL, said colors can be well represented within $\mathbb{R}$, hence there exists an injective $\phi$ in Equation (4) with $d = 1$. This completes the proof. □

We now show Theorem 1. Specifically, we show the following two propositions from which Theorem 1 follows.

**Proposition 7.** *Let $G = (V(G), E(G), \ell)$ be a node-labeled graph with $n$ nodes and $\boldsymbol{F} \in \mathbb{R}^{n \times p}$ be a node feature matrix consistent with $\ell$. Then for all $t \geq 0$, there exists a parametrization of the ET such that*

$$C_t^{2,F}(\boldsymbol{v}) = C_t^{2,F}(\boldsymbol{w}) \impliedby \boldsymbol{X}^{(t)}(\boldsymbol{v}) = \boldsymbol{X}^{(t)}(\boldsymbol{w}),$$

*for all pairs of 2-tuples $\boldsymbol{v}$ and $\boldsymbol{w} \in V(G)^2$.*

*Proof.* We begin by stating that our domain is compact since the ET merely operates on at most $n$ possible node features in $\boldsymbol{F}$ and binary edge features in $\boldsymbol{E}$, and at each iteration there exist at most $n^2$ distinct 2-FWL colors. We prove our statement by induction over iteration $t$. For the base case, we can simply invoke Proposition 6 since our input tokens are constructed according to Equation (4). Nonetheless, we show a possible initialization of the tokenization that is consistent with Equation (4) that we will use in the induction step.

From Proposition 6, we know that the color representation of a tuple can be represented in $\mathbb{R}$. We denote the color representation of a tuple $\boldsymbol{u} = (i, j)$ at iteration $t$ as $\boldsymbol{T}^{(t)}(\boldsymbol{u})$ and $\boldsymbol{T}_{ij}^{(t)}$ interchangeably. We choose a $\phi$ in Equation (4) such that for each $\boldsymbol{u} = (i, j)$

$$\boldsymbol{X}_{ij}^{(0)} = \begin{bmatrix} \boldsymbol{T}_{ij}^{(0)} & \left(\boldsymbol{T}_{ij}^{(0)}\right)^{n^2} \end{bmatrix} \in \mathbb{R}^2,$$

where we store the tuple features, one with exponent 1 and once with exponent $n^2$ and where $\boldsymbol{T}_{ij}^{(0)} \in \mathbb{R}$ and $\left(\boldsymbol{T}_{ij}^{(0)}\right)^{n^2} \in \mathbb{R}$. We choose color representations $\boldsymbol{T}_{ij}^{(0)}$ as follows. First, we define an injective function $f_t : V(G)^2 \to [n^2]$ that maps each 2-tuple $\boldsymbol{u}$ to a number in $[n^2]$ unique for its 2-FWL color $C_t^{2,\mathrm{F}}(\boldsymbol{u})$ at iteration $t$. Note that $f_t$ can be injective because there can at most be $[n^2]$ unique numbers under the 2-FWL. We will use $f_t$ to map each tuple color under the 2-FWL to a unique $n$-ary number. We then choose $\phi$ in Equation (4) such that for each $(i, j) \in V(G)^2$,

$$\left|\left|\boldsymbol{T}_{ij}^{(0)} - n^{-f_0(i,j)}\right|\right|_F < \epsilon_0,$$

for all $\epsilon_0 > 0$, by the universal function approximation theorem, which we can invoke since our domain is compact. We will use $\left(\boldsymbol{T}_{ij}^{(0)}\right)^{n^2}$ in the induction step; see below.

For the induction, we assume that

$$C_{t-1}^{2,\mathrm{F}}(\boldsymbol{v}) = C_{t-1}^{2,\mathrm{F}}(\boldsymbol{w}) \impliedby \boldsymbol{T}^{(t-1)}(\boldsymbol{v}) = \boldsymbol{T}^{(t-1)}(\boldsymbol{w})$$

and that

$$\left|\left|\boldsymbol{T}_{ij}^{(t-1)} - n^{-f_{t-1}(i,j)}\right|\right|_F < \epsilon_{t-1},$$

for all $\epsilon_{t-1} > 0$ and $(i, j) \in V(G)^2$. We then want to show that there exists a parameterization of the $t$-th layer such that

$$C_t^{2,\mathrm{F}}(\boldsymbol{v}) = C_t^{2,\mathrm{F}}(\boldsymbol{w}) \impliedby \boldsymbol{T}^{(t)}(\boldsymbol{v}) = \boldsymbol{T}^{(t)}(\boldsymbol{w}) \tag{7}$$

and that

$$\left|\left|\boldsymbol{T}_{ij}^{(t)} - n^{-f_t(i,j)}\right|\right|_F < \epsilon_t,$$

for all $\epsilon_t > 0$ and $(i,j) \in V(G)^2$. Clearly, if this holds for all $t$, then the proof statement follows. Thereto, we show that the ET updates the tuple representation of tuple $(j,m)$ as

$$\boldsymbol{T}_{jm}^{(t)} = \mathsf{FFN}\Big(\boldsymbol{T}_{jm}^{(t-1)} + \frac{\beta}{n}\sum_{l=1}^{n} \boldsymbol{T}_{jl}^{(t-1)} \cdot \Big(\boldsymbol{T}_{lm}^{(t-1)}\Big)^{n^2}\Big), \tag{8}$$

for an arbitrary but fixed $\beta$. We first show that then, Equation (7) holds. Afterwards we show that the ET can indeed compute Equation (8). To show the former, note that for two 2-tuples $(j,l)$ and $(l,m)$,

$$n^{-n^2} \cdot n^{-f_{t-1}(j,l)} \cdot \Big(n^{-f_{t-1}(l,m)}\Big)^{n^2} = n^{-(n^2 + f_{t-1}(j,l) + n^2 \cdot f_{t-1}(l,m))},$$

is unique for the pair of colors

$$\big(C_t^{2,\mathrm{F}}((j,l)), C_t^{2,\mathrm{F}}((l,m))\big)$$

where $n^{-n^2}$ is a constant normalization term we will later introduce with $\frac{\beta}{n}$. Note further, that we have

$$\left|\left|\boldsymbol{T}_{jl}^{(t-1)} \cdot \Big(\boldsymbol{T}_{lm}^{(t-1)}\Big)^{n^2} - n^{-(n^2 + f_{t-1}(j,l) + n^2 \cdot f_{t-1}(l,m))}\right|\right|_F < \delta_{t-1},$$

for all $\delta_{t-1} > 0$. Further, $n^{-(f_{t-1}(j,l) + n^2 \cdot f_{t-1}(l,m))}$ is still an $m$-ary number with $m = n$. As a result, we can set $\beta = n^{-n^2+1}$ and invoke Lemma 4 to obtain that

$$\frac{\beta}{n} \cdot \sum_{l=1}^{n} n^{-(f_{t-1}(j,l) + n^2 \cdot f_{t-1}(l,m))} = \sum_{l=1}^{n} n^{-(n^2 + f_{t-1}(j,l) + n^2 \cdot f_{t-1}(l,m))},$$

is unique for the multiset of colors

$$\{\!\{(C_{t-1}^{2,\mathrm{F}}((l,m)), C_{t-1}^{2,\mathrm{F}}((j,l))) \mid l \in V(G)\}\!\},$$

and we have that

$$\left|\left|\frac{\beta}{n}\sum_{l=1}^{n} \boldsymbol{T}_{jl}^{(t-1)} \cdot \Big(\boldsymbol{T}_{lm}^{(t-1)}\Big)^{n^2} - \sum_{l=1}^{n} n^{-(n^2 + f_{t-1}(j,l) + n^2 \cdot f_{t-1}(l,m))}\right|\right|_F < \gamma_{t-1},$$

for all $\gamma_{t-1} > 0$. Finally, we define

$$A \coloneqq \Big\{n^{-f_{t-1}(j,m)} \mid (j,m) \in V(G)^2\Big\}$$
$$B \coloneqq \Big\{\frac{\beta}{n} \cdot \sum_{l=1}^{n} n^{-(f_{t-1}(j,l) + n^2 \cdot f_{t-1}(l,m))} \mid (j,m) \in V(G)^2\Big\}.$$

Further, because we multiply with $\frac{\beta}{n}$, we have that

$$\min_{x \in A} x > \max_{y \in B} y$$

and as a result, by Lemma 5,

$$n^{-f_{t-1}(j,m)} + \frac{\beta}{n} \cdot \sum_{l=1}^{n} n^{-(f_{t-1}(j,l) + n^2 \cdot f_{t-1}(l,m))}$$

is unique for the pair

$$\big(C_{t-1}^{2,\mathrm{F}}((j,m)), \{\!\{(C_{t-1}^{2,\mathrm{F}}((l,m)), C_{t-1}^{2,\mathrm{F}}((j,l))) \mid l \in V(G)\}\!\}\big)$$

and consequently for color $C_t^{2,\mathrm{F}}((j,m))$ at iteration $t$. Further, we have that

$$\left|\left|\boldsymbol{T}_{jm}^{(t-1)} + \frac{\beta}{n}\sum_{l=1}^{n} \boldsymbol{T}_{jl}^{(t-1)} \cdot \Big(\boldsymbol{T}_{lm}^{(t-1)}\Big)^{n^2} - n^{-f_{t-1}(j,m)} + \frac{\beta}{n} \cdot \sum_{l=1}^{n} n^{-(f_{t-1}(j,l) + n^2 \cdot f_{t-1}(l,m))}\right|\right|_F < \tau_{t-1},$$

for all $\tau_{t-1} > 0$. Finally, since our domain is compact, we can invoke universal function approximation with $\mathsf{FFN}$ in Equation (8) to obtain

$$\left|\left|\boldsymbol{T}_{jm}^{(t)} - n^{-f_t(j,m)}\right|\right|_F < \epsilon_t,$$

for all $\epsilon_t > 0$. Further, because $n^{-f_t(j,m)}$ is unique for each unique color $C_t^{2,\mathrm{F}}((j,m))$, Equation (7) follows.

It remains to show that the ET can indeed compute Equation (8). To this end, we will require a single transformer head in each layer. Specifically, we want this head to compute

$$h_1(\boldsymbol{X}^{(t-1)})_{jm} = \frac{\beta}{n}\sum_{l=1}^{n} \boldsymbol{T}_{jl}^{(t-1)} \cdot \left(\boldsymbol{T}_{lm}^{(t-1)}\right)^{n^2}. \tag{9}$$

Now, recall the definition of the Edge Transformer head at tuple $(j,m)$ as

$$h_1(\boldsymbol{X}^{(t-1)})_{jm} \coloneqq \sum_{l=1}^{n} \alpha_{jlm} \boldsymbol{V}_{jlm}^{(t-1)},$$

where

$$\alpha_{jlm} \coloneqq \underset{l\in[n]}{\mathsf{softmax}}\Big(\frac{1}{\sqrt{d_k}} \boldsymbol{X}_{jl}^{(t-1)} \boldsymbol{W}^Q (\boldsymbol{X}_{lm}^{(t-1)} \boldsymbol{W}^K)^T\Big)$$

with

$$\boldsymbol{V}_{jlm}^{(t-1)} \coloneqq \boldsymbol{X}_{jl}^{(t-1)} \begin{bmatrix} \boldsymbol{W}_1^{V_1} \\ \boldsymbol{W}_2^{V_1} \end{bmatrix} \odot \boldsymbol{X}_{lm}^{(t-1)} \begin{bmatrix} \boldsymbol{W}_1^{V_2} \\ \boldsymbol{W}_2^{V_2} \end{bmatrix}$$

and by the induction hypothesis above,

$$\begin{aligned} \boldsymbol{X}_{jl}^{(t-1)} &= \begin{bmatrix} \boldsymbol{T}_{jl}^{(t-1)} & \left(\boldsymbol{T}_{jl}^{(t-1)}\right)^{n^2} \end{bmatrix} \\ \boldsymbol{X}_{lm}^{(t-1)} &= \begin{bmatrix} \boldsymbol{T}_{lm}^{(t-1)} & \left(\boldsymbol{T}_{lm}^{(t-1)}\right)^{n^2} \end{bmatrix}, \end{aligned}$$

where we expanded sub-matrices. Specifically, $\boldsymbol{W}_1^{V_1}, \boldsymbol{W}_1^{V_2}, \boldsymbol{W}_2^{V_1}, \boldsymbol{W}_2^{V_2} \in \mathbb{R}^{\frac{d}{2}\times d}$. We then set

$$\begin{aligned} \boldsymbol{W}^Q &= \boldsymbol{W}^K = \boldsymbol{0} \\ \boldsymbol{W}_1^{V_1} &= [\beta \boldsymbol{I} \quad \boldsymbol{0}] \\ \boldsymbol{W}_2^{V_1} &= [\boldsymbol{0} \quad \boldsymbol{0}] \\ \boldsymbol{W}_1^{V_2} &= [\boldsymbol{0} \quad \boldsymbol{I}] \\ \boldsymbol{W}_2^{V_2} &= [\boldsymbol{0} \quad \boldsymbol{0}]. \end{aligned}$$

Here, $\boldsymbol{W}^Q$ and $\boldsymbol{W}^K$ are set to zero to obtain uniform attention scores. Note that then for all $j, l, k$, $\alpha_{jlm} = \frac{1}{n}$, due to normalization over $l$, and we end up with Equation (9) as

$$h_1(\boldsymbol{X}^{(t-1)})_{jm} = \frac{1}{n}\sum_{l=1}^{n} \boldsymbol{V}_{jlm}^{(t-1)}$$

where

$$\begin{aligned} \boldsymbol{V}_{jlm}^{(t-1)} &= \left[\boldsymbol{T}_{jl}^{(t-1)} \cdot \beta \boldsymbol{I} + \left(\boldsymbol{T}_{jl}^{(t-1)}\right)^{n^2} \cdot \boldsymbol{0} \quad \boldsymbol{0}\right] \odot \left[\boldsymbol{T}_{lm}^{(t-1)} \cdot \boldsymbol{0} + \left(\boldsymbol{T}_{lm}^{(t-1)}\right)^{n^2} \cdot \boldsymbol{I} \quad \boldsymbol{0}\right] \\ &= \beta \cdot \left[\boldsymbol{T}_{jl}^{(t-1)} \cdot \left(\boldsymbol{T}_{lm}^{(t-1)}\right)^{n^2} \quad \boldsymbol{0}\right]. \end{aligned}$$

We now conclude our proof as follows. Recall that the Edge Transformer layer computes the final representation $\boldsymbol{X}^{(t)}$ as

$$\begin{aligned}
\boldsymbol{X}^{(t)}_{jm} &= \mathsf{FFN}\left(\boldsymbol{X}^{(t-1)}_{jm} + h_1(\boldsymbol{X}^{(t-1)})_{jm}\boldsymbol{W}^O\right) \\
&= \mathsf{FFN}\left(\left[\boldsymbol{T}^{(t-1)}_{jm} \quad \left(\boldsymbol{T}^{(t-1)}_{jm}\right)^{n^2}\right] + \frac{\beta}{n}\sum_{l=1}^{n}\left[\boldsymbol{T}^{(t-1)}_{jl}\cdot\boldsymbol{T}^{(t-1)}_{lm} \quad \mathbf{0}\right]\boldsymbol{W}^O\right) \\
&\underset{\boldsymbol{W}^O:=\boldsymbol{I}}{=} \mathsf{FFN}\left(\left[\boldsymbol{T}^{(t-1)}_{jm} \quad \left(\boldsymbol{T}^{(t-1)}_{jm}\right)^{n^2}\right] + \left[\tfrac{\beta}{n}\textstyle\sum_{l=1}^{n}\boldsymbol{T}^{(t-1)}_{jl}\cdot\boldsymbol{T}^{(t-1)}_{lm} \quad \mathbf{0}\right]\right) \\
&= \mathsf{FFN}\left(\left[\boldsymbol{T}^{(t-1)}_{jm} + \tfrac{\beta}{n}\textstyle\sum_{l=1}^{n}\boldsymbol{T}^{(t-1)}_{jl}\cdot\boldsymbol{T}^{(t-1)}_{lm} \quad \left(\boldsymbol{T}^{(t-1)}_{jm}\right)^{n^2}\right]\right) \\
&\underset{Eq.8}{=} \mathsf{FFN}\left(\left[\boldsymbol{T}^{(t)}_{jm} \quad \left(\boldsymbol{T}^{(t-1)}_{jm}\right)^{n^2}\right]\right)
\end{aligned}$$

for some $\mathsf{FFN}$. Note that the above derivation only modifies the terms inside the parentheses and is thus independent of the choice of $\mathsf{FFN}$. We have thus shown that the ET can compute Equation (8).

To complete the induction, let $f\colon \mathbb{R}^2 \to \mathbb{R}^2$ be such that

$$f\left(\left[\boldsymbol{T}^{(t)}_{jm} \quad \left(\boldsymbol{T}^{(t-1)}_{jm}\right)^{n^2}\right]\right) = \left[\boldsymbol{T}^{(t)}_{jm} \quad \left(\boldsymbol{T}^{(t)}_{jm}\right)^{n^2}\right].$$

Since our domain is compact, $f$ is continuous, and hence we can choose $\mathsf{FFN}$ to approximate $f$ arbitrarily close. This completes the proof. □

Next, we show the other direction of Theorem 1.

**Proposition 8.** *For all parametrizations of the ET, there exists a* $\mathsf{recolor}$ *function such that for all* $n > 0$ *and all node-labeled graphs* $G = (V(G), E(G), \ell)$ *with* $n$ *nodes and node feature matrix* $\boldsymbol{F} \in \mathbb{R}^{n\times p}$ *consistent with* $\ell$ *and all* $t \geq 0$ *such that*

$$C^{2,F}_t(\boldsymbol{v}) = C^{2,F}_t(\boldsymbol{w}) \implies \boldsymbol{X}^{(t)}(\boldsymbol{v}) = \boldsymbol{X}^{(t)}(\boldsymbol{w}),$$

*for all pairs of* 2*-tuples* $\boldsymbol{v}$ *and* $\boldsymbol{w} \in V(G)^2$.

*Proof.* We begin by stating that our domain is compact since the ET merely operates on at most $n$ possible node features in $\boldsymbol{F}$ and binary edge features in $\boldsymbol{E}$ and at each iteration $t \geq 0$ there exist at most $n^2$ distinct embeddings $\boldsymbol{X}^{(t)}(i,j)$, for all $i, j \in V(G)$. Further, we recall iteration $t > 0$ of the 2-$\mathsf{FWL}$ as computing

$$C^{2,\mathrm{F}}_t(i,j) = \mathsf{recolor}\Big(\big(C^{2,\mathrm{F}}_{t-1}(i,j), \{\!\{(C^{2,\mathrm{F}}_{t-1}(i,l), C^{2,\mathrm{F}}_{t-1}(l,j)) \mid l \in V(G)\}\!\}\big)\Big)$$

for all pairs of nodes $i, j \in V(G)$, where $\mathsf{recolor}$ is an invertible function mapping tuples of colors to colors.

We prove the statement by induction over $t$. For $t = 0$, the function $\mathsf{recolor}$ just maps the initial embedding $\boldsymbol{X}^{(0)}_{ij}$ to a color $C^{2,\mathrm{F}}_0(i,j)$ unique for each unique value of $\boldsymbol{X}^{(0)}_{ij}$. Since there at most $n^2$ possible embeddings $\boldsymbol{X}^{(0)}_{ij}$, such a mapping always exists and is bijective. We denote this mapping $\tau_0$ and have that

$$\tau_0(\boldsymbol{X}^{(0)}_{ij}) = C^{2,\mathrm{F}}_0(i,j)$$

for all pairs of nodes $i, j \in V(G)$. We will use $\tau_0$ in the induction step. Clearly, we have that

$$C^{2,\mathrm{F}}_0(\boldsymbol{v}) = C^{2,\mathrm{F}}_0(\boldsymbol{w}) \implies \boldsymbol{X}^{(0)}(\boldsymbol{v}) = \boldsymbol{X}^{(0)}(\boldsymbol{w}),$$

for all pairs of 2-tuples $\boldsymbol{v}$ and $\boldsymbol{w} \in V(G)^2$.

For $t > 0$, we describe recolor step-by-step. By the induction hypothesis we have that

$$C^{2,\mathrm{F}}_{t-1}(\boldsymbol{v}) = C^{2,\mathrm{F}}_{t-1}(\boldsymbol{w}) \Longrightarrow \boldsymbol{X}^{(t-1)}(\boldsymbol{v}) = \boldsymbol{X}^{(t-1)}(\boldsymbol{w}),$$

for all pairs of 2-tuples $\boldsymbol{v}$ and $\boldsymbol{w} \in V(G)^2$. Further, we have an invertible mapping $\tau_{t-1}$ such that

$$\tau_{t-1}(\boldsymbol{X}^{(t-1)}_{ij}) = C^{2,\mathrm{F}}_{t-1}(i,j),$$

for all pairs of nodes $i, j \in V(G)$. First, recolor decodes its input to obtain colors $C^{2,\mathrm{F}}_{t-1}(i,j)$ and multiset $\{\!\{(C^{2,\mathrm{F}}_{t-1}(i,l), C^{2,\mathrm{F}}_{t-1}(l,j)) \mid l \in V(G)\}\!\}$, which is possible since recolor is invertible. We define

$$\begin{aligned}\hat{\boldsymbol{X}}^{(t-1)}_{ij} &\coloneqq \tau^{-1}_{t-1}(C^{2,\mathrm{F}}_{t-1}(i,j)) \\ &= \boldsymbol{X}^{(t-1)}_{ij}\end{aligned}$$

and

$$\begin{aligned}\hat{\boldsymbol{Z}}^{(t-1)}_{ij} &\coloneqq \{\!\{(\tau^{-1}_{t-1}(C^{2,\mathrm{F}}_{t-1}(i,l)), \tau^{-1}_{t-1}(C^{2,\mathrm{F}}_{t-1}(l,j))) \mid l \in V(G)\}\!\} \\ &= \{\!\{(\boldsymbol{X}^{(t-1)}_{il}, \boldsymbol{X}^{(t-1)}_{lj}) \mid l \in V(G)\}\!\},\end{aligned}$$

where recolor uses the inverse of $\tau^{-1}_{t-1}$ to decode $C^{2,\mathrm{F}}_{t-1}(i,j)$ into its corresponding ET embedding and the multiset further into a multiset of ET embeddings at iteration $t-1$. Now, recolor computes

$$\boldsymbol{X}^{(t)}_{ij} = \mathsf{FFN}\Big(\boldsymbol{X}^{(t-1)}_{ij} + \mathsf{TriAttention}\big(\boldsymbol{X}^{(t-1)}_{ij}\big)\Big),$$

where the first summand in the FFN is obtained from $\hat{\boldsymbol{X}}^{(t-1)}_{ij}$ and the second summand in the FFN is obtained from $\hat{\boldsymbol{Z}}^{(t-1)}_{ij}$ since $\mathsf{TriAttention}(\boldsymbol{X}^{(t-1)}_{ij})$ is a function of $\hat{\boldsymbol{Z}}^{(t-1)}$.

To conclude the induction step, recolor maps $\boldsymbol{X}^{(t)}_{ij}$ to a color $C^{2,\mathrm{F}}_{t}(i,j)$ unique for each unique value of $\boldsymbol{X}^{(t)}_{ij}$. Since there at most $n^2$ possible embeddings $\boldsymbol{X}^{(t)}_{ij}$, such a mapping always exists and is bijective. We denote this mapping

$$\tau_t(\boldsymbol{X}^{(t)}_{ij}) \coloneqq C^{2,\mathrm{F}}_{t}(i,j).$$

This concludes the induction and hence, the proof. □

Note that unlike the result in Proposition 7, the above result is uniform, in that the concrete choice of recolor does not depend on the graph size $n$. Finally, Theorem 1 follows from Proposition 7 and Proposition 8.