An end-to-end attention-based approach for learning on graphs

David Buterez[1], Jon Paul Janet[2], Dino Oglic[3], and Pietro Liò[1]

[1]Department of Computer Science and Technology, University of Cambridge, Cambridge, UK
[2]Molecular AI, BioPharmaceuticals R&D, AstraZeneca, Gothenburg, Sweden
[3]Centre for AI, BioPharmaceuticals R&D, AstraZeneca, Cambridge, UK

Abstract

There has been a recent surge in transformer-based architectures for learning on graphs, mainly motivated by attention as an effective learning mechanism and the desire to supersede handcrafted operators characteristic of message passing schemes. However, concerns over their empirical effectiveness, scalability, and complexity of the pre-processing steps have been raised, especially in relation to much simpler graph neural networks that typically perform on par with them across a wide range of benchmarks. To tackle these shortcomings, we consider graphs as sets of edges and propose a purely attention-based approach consisting of an encoder and an attention pooling mechanism. The encoder vertically interleaves masked and vanilla self-attention modules to learn an effective representations of edges, while allowing for tackling possible misspecifications in input graphs. Despite its simplicity, the approach outperforms fine-tuned message passing baselines and recently proposed transformer-based methods on more than 70 node and graph-level tasks, including challenging long-range benchmarks. Moreover, we demonstrate state-of-the-art performance across different tasks, ranging from molecular to vision graphs, and heterophilous node classification. The approach also outperforms graph neural networks and transformers in transfer learning settings, and scales much better than alternatives with a similar performance level or expressive power.

1 Introduction

We investigate empirically the potential of a purely attention-based approach for learning effective representations of graph structured data. Typically, learning on graphs is modelled as message passing – an iterative process that relies on a message function to aggregate information from a given node's neighbourhood and an update function to incorporate the encoded message into the output representation of the node. The resulting graph neural networks (GNNs) typically stack multiple such layers to learn node representations based on vertex rooted sub-trees, essentially mimicking the one-dimensional Weisfeiler–Lehman (1-WL) graph isomorphism test [1, 2]. Variations of message passing have been applied effectively in different fields such as life sciences [3–8], electrical engineering [9], and weather prediction [10].

Despite the overall success and wide adoption of graph neural networks, several practical challenges have been identified over time. While the message passing framework is highly flexible, the design of new layers is a challenging research problem where improvements take years to achieve and often rely on hand-crafted operators. This is particularly the case for general purpose graph neural networks that do not exploit additional input modalities such as atomic coordinates. For instance, principal neighbourhood aggregation (PNA) is regarded as one of the most powerful message passing layers [11], but it is built using a collection of manually selected neighbourhood aggregation functions, requires a dataset degree histogram which must be pre-computed prior to learning, and further uses manually selected degree scaling. The nature of message passing also imposes certain limitations which have shaped the majority of the literature. One of the most prominent examples is the readout function used to combine node-level features into a single graph-level representation, and which is required to be permutation invariant with respect to the node order. Thus, the default choice for graph neural networks and even graph transformers remains a simple, non-learnable function such as sum, mean, or max [12–14]. Limitations of this approach have been identified by Wagstaff et al. [15], who have shown that simple readout functions might require complex item embedding functions that are difficult to learn using standard neural networks. Additionally, graph neural networks have shown limitations in terms of over-smoothing [16], linked to node representations becoming similar with increased depth, and over-squashing [17] due to information compression through bottleneck edges. The proposed solutions typically take the form of message regularisation schemes [18–20]. However, there is generally no consensus on the right architectural choices for building effective deep message passing neural networks. Transfer learning and strategies like pre-training and fine-tuning are also less ubiquitous in graph neural networks because of modest or ambiguous benefits, as opposed to large language models [21].

The attention mechanism [22] is one of the main sources of innovation within graph learning, either by directly incorporating attention within message passing [23, 24], by formulating graph learning as a language processing task [25, 26], or by combining vanilla GNN layers with attention layers [12, 13, 27, 28]. However, several concerns have been raised regarding the performance, scalability, and complexity of such methods.

Performance-wise, recent reports indicate that sophisticated graph transformers underperform compared to simple but tuned GNNs [29, 30]. This line of work highlights the importance of empirically evaluating new methods relative to strong baselines. Separately, recent graph transformers have focused on increasingly more complex helper mechanisms, such as computationally expensive pre-processing and learning steps [25], various different encodings (e.g., positional, structural, and relational) [25–27, 31], inclusion of virtual nodes and edges [25, 28], conversion of the problem to natural language processing [25, 26], and other non-trivial graph transformations [28, 31]. These complications can significantly increase the computational requirements, reducing the chance of being widely adopted and replacing GNNs.

Motivated by the effectiveness of attention as a learning mechanism and recent advances in efficient and exact attention, we introduce an end-to-end attention-based architecture for learning on graphs that is simple to implement, scalable, and achieves state-of-the-art results. The proposed architecture considers graphs as sets of edges, leveraging an encoder that interleaves masked and self-attention mechanisms to learn effective representations. The attention-based pooling component mimics the functionality of a readout function and it is responsible for aggregating the edge-level features into a permutation invariant graph-level representation. The masked attention mechanism allows for learning effective edge representations originating from the graph connectivity and the combination with self-attention layers vertically allows for expanding on this information while having a strong prior. Masking can, thus, be seen as leveraging specified relational information and its vertical combination with self-attention as a means to overcome possible misspecification of the input graph. The masking operator is injected into the pairwise attention weight matrix and allows only for attention between linked primitives. For a pair of edges, the connectivity translates to having a shared node between them. We focus primarily on learning through edge sets due to empirically high performance, and refer to our architecture as edge-set attention (ESA). We also demonstrate that the overall architecture is effective for propagating information across nodes through dedicated node-level benchmarks. The ESA architecture is general purpose, in the sense that it only relies on the graph structure and possibly node and edge features, and it is not restricted to any particular domain. Furthermore, ESA does not use positional, structural, relative, or similar encodings, it does not encode graph structures as tokens or other language (sequence) specific concepts, and it does not require any pre-computations.

Despite its apparent simplicity, ESA-based learning overwhelmingly outperforms strong and tuned GNN baselines and much more involved transformer-based models. Our evaluation is extensive, totalling 70 datasets and benchmarks from different domains such as quantum mechanics, molecular docking, physical chemistry, biophysics, bioinformatics, computer vision, social networks, functional call graphs, and synthetic graphs. At the node level, we include both homophilous and heterophilous graph tasks, as well as shortest path problems, as these require modelling long-range interactions. Beyond supervised learning tasks, we explore the potential for transfer learning in the context of drug discovery and quantum mechanics [32] and show that ESA is a viable transfer learning strategy compared to vanilla GNNs and graph transformers.

2 Related Work

An attention mechanism that mimics message passing and limits the attention computations to neighbouring nodes has been first proposed in GAT [23]. We consider masking as an abstraction of the GAT attention operator that allows for building general purpose relational structures between items in a set (e.g., k-hop neighbourhoods or *conformational* masks extracted from 3D molecular structures). The attention mechanism in GAT is implemented as a single linear projection matrix that does not explicitly distinguish between keys, queries, and values as in standard dot product attention and an additional linear layer after concatenating the representations of connected nodes, along with a non-linearity. This type of simplified attention has been labelled by subsequent work as *static* and was shown to have limited expressive power [24]. Brody et al. instead proposed *dynamic* attention, a simple reordering of operations in GAT, resulting in a more expressive GATv2 model [24] – however, at the price of doubling the parameter count and the corresponding memory consumption. A high-level overview of GAT in the context of masked attention is provided in Figure 1, along with the main differences to the proposed architecture. Similar to GAT, several adaptations of the original scaled dot product attention have been proposed for graphs [33, 34], where the focus was on defining an attention mechanism constrained by the node connectivity and replacing the positional encodings of the original transformer model with more appropriate graph alternatives, such as Laplacian eigenvectors. These approaches, while interesting and forward-looking, did not convincingly outperform simple GNNs. Building up on this line of work, an architecture that can be seen as an instance of masked transformers has been proposed in SAN [35], illustrated in Figure 2. Attention coefficients there are defined as a convex combination of the scores (controlled by hyper-parameter γ) associated with the original graph and its complement. Min et al. [36] have also considered a masking mechanism for standard transformers. However, the graph structure itself is not used directly in the masking process as the devised masks correspond to four types of prior interaction graphs (induced, similarity, cross neighbourhood, and complete sub-graphs), acting as an inductive bias. Furthermore, the method was not designed for general purpose graph learning as it relies on helper mechanisms such as neighbour sampling and a heterogeneous information network.
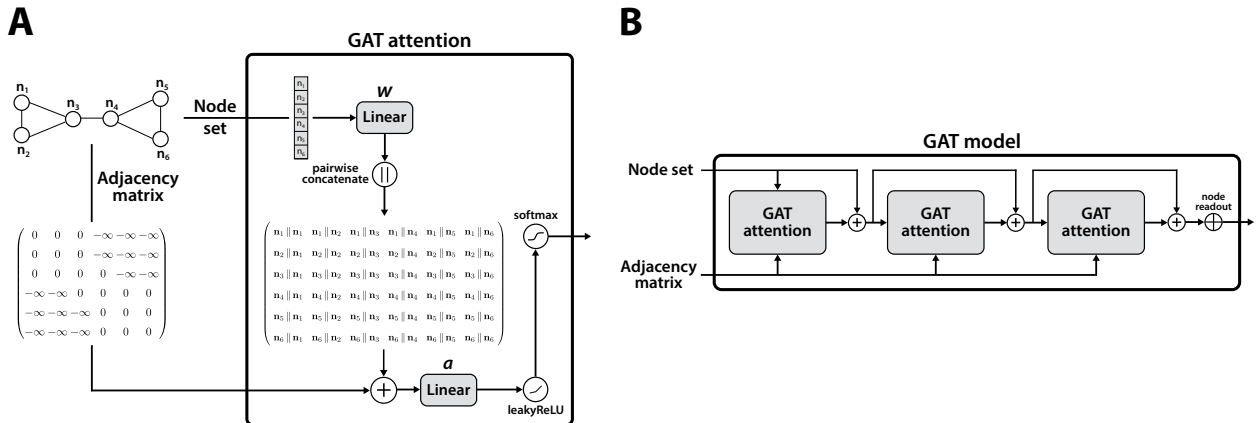
Figure 1: A high level overview of the GAT message passing algorithm [23]. **Panel A:** A GAT layer receives node representations and the adjacency matrix as inputs. First a projection matrix is applied to all the nodes, which are then concatenated pairwise and passed through another linear layer, followed by a non-linearity. The final attention score is computed by the softmax function. **Panel B:** A GAT model stacks multiple GAT layers and uses a readout function over nodes to generate a graph-level representation. Residual connections are also illustrated as a modern enhancement of GNNs (not included in the original approach). **GAT vs ESA:** Our masked self-attention module relies on masking within classical scaled dot product attention which comes with different projection matrices for keys, queries, and values. Additionally, we wrap the masked scaled dot product attention with layer normalization and skip connections, the output of which is passed through an MLP. The latter is not done classically in GAT as part of its attention modules nor in the overall architecture. In contrast to GAT that operates over nodes, our masking operator is defined over sets of edges that are connected if they share a node in common. When it comes to the readout, GAT uses sum, mean or max and aggregates over nodes whereas our architecture relies on pooling by multi-head attention and aggregation over learned representations of seed vectors inherent to that module. In terms of the encoder that is responsible for learning representations of set items, ours interleaves masked and self-attention layers vertically whereas in GATs the stacked layers are entirely based on neighbourhood attention.

Recent trends in learning on graphs are dominated by architectures based on standard self-attention layers as the only learning mechanism, with a significant amount of effort put into representing the graph structure exclusively through positional and structural encodings. Graphormer [25] is one of the most prominent approaches from this class. It comes with an involved and computation-heavy suite of pre-processing steps, involving a centrality, spatial, and edge encoding. For instance, spatial encodings rely on the Floyd–Warshall algorithm that has cubic time complexity in the number of nodes and quadratic memory complexity. Also, the model employs a virtual mean-readout node that is connected to all other nodes in the graph. While Graphormer has originally been evaluated only on four datasets, the results were promising relative to GNNs and SAN. Another related approach is the Tokenized Graph Transformer (TokenGT) [26], which treats all the nodes and edges as independent tokens. To adapt sequence learning to the graph domain, TokenGT encodes the graph information using node identifiers derived from orthogonal random features or Laplacian eigenvectors, and learnable type identifiers for nodes and edges. TokenGT is provably more expressive than standard GNNs, and can approximate k-WL tests with the appropriate architecture (number of layers, adequate pooling, etc.). However, this theoretical guarantee holds only with the use of positional encodings that typically break the permutation invariance over nodes that is required for consistent predictions over the same graph presented with a different node order. The main strength of TokenGT is its theoretical expressiveness, as it has only been evaluated on a single dataset where it did not outperform Graphormer.

A route involving a hybrid between classical transformers and message passing has been pursued in the GraphGPS framework [37], which combines message passing and transformer layers. As in previous works, GraphGPS puts a large emphasis on different types of encodings, proposing and analysing positional and structural encodings, further divided into local, global, and relative encodings. Exphormer [28] is an evolution of GraphGPS that adds virtual global nodes and sparse attention based on expander graphs. While effective, such frameworks do still rely on message passing and are thus not purely attention-based solutions. Limitations include dependence on approximations (Performer [38] for both, expander graphs for Exphormer), and decreased performance when encodings (GraphGPS) or special nodes (Exphormer) are removed. Notably, Exphormer is the first approach from this class to consider custom attention patterns given by node neighbourhoods.

3 Methods

In this section, we provide a high-level overview of our architecture and describe the masked attention module that allows for information propagation across edges as primitives. An alternative implementation for the more traditional node-based propagation is also presented. The section starts with a formal definition of the masked self-attention mechanism and then proceeds to describe our end-to-end attention-based approach for learning on graphs, involving the encoder and pooling components (illustrated in Figure 3).
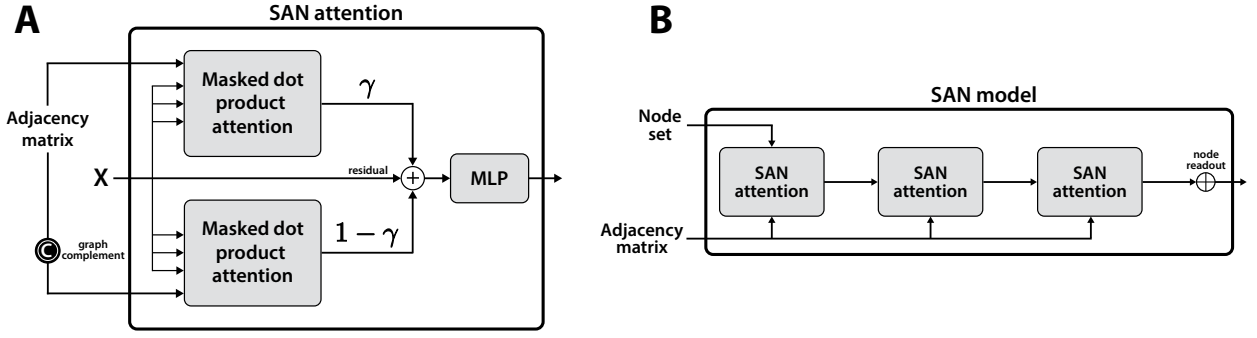
Figure 2: A high level overview of the SAN architecture [35]. **Panel A:** Two independent attention modules with a tied/shared value projection matrix are used for the input graph and its complement. The outputs of these two modules are convexly combined using a hyperparameter γ before being residually added to the input. **Panel B:** The overall SAN architecture that stacks multiple SAN layers without residual connections, and uses a standard readout function over nodes. **SAN vs ESA:** What is truly different in relation to this architecture is horizontal versus vertical combination of masked and self-attention. While it is possible to set $\gamma = 0$ for some layers and $\gamma = 1$ for others, this avenue has never been explored in SAN and it would still result in a slightly different encoder as the query and key projections are different for the input graph and its complement. The empirical analysis in [35] also provides no insights relative to vertical combination of masked and self-attention, nor the impact of attention pooling which is a differentiating factor and an important part of our architecture.
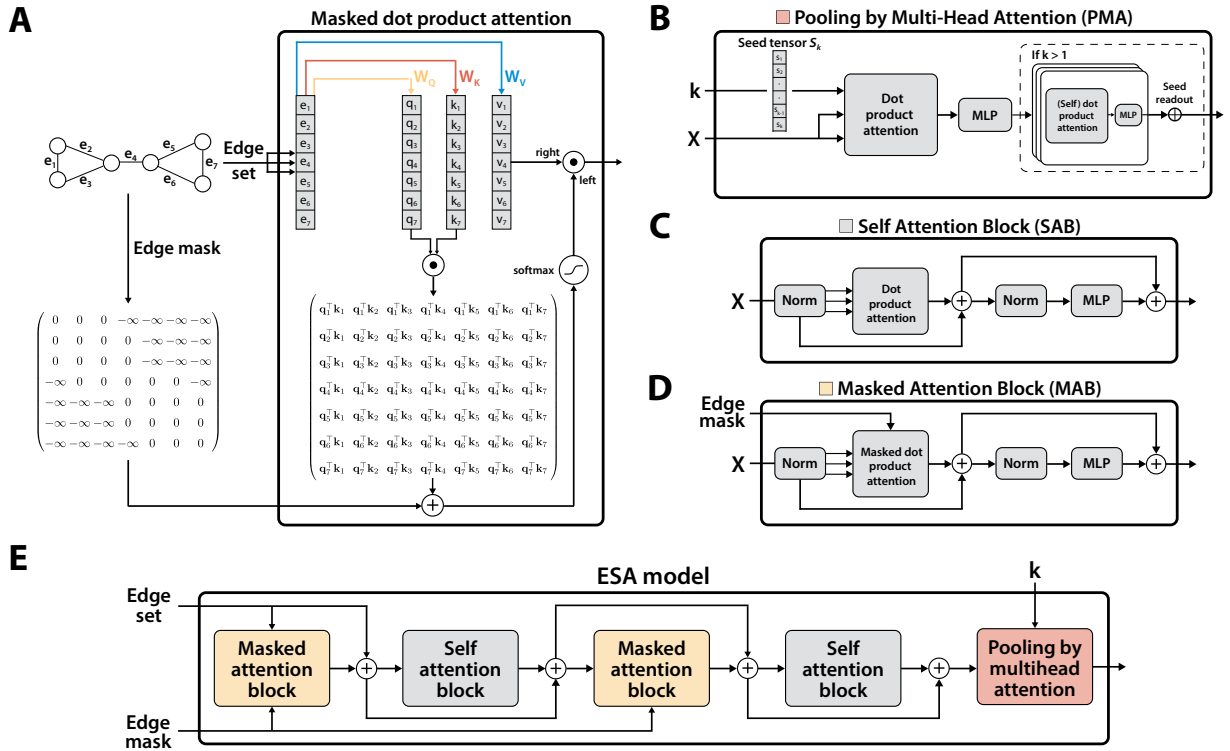

Figure 3: A high-level overview of Edge Set Attention and its main building blocks. **Panel A:** An illustration of masked scaled dot product attention with edge inputs. Edge features derived from the edge set are processed by query, key, and value projections as in standard attention. An additive edge mask, derived as in Algorithm 1, is applied prior to the softmax operator. A mask value of 0 leaves the attention score unchanged, while a large negative value completely discards that attention score. **Panel B:** The Pooling by Multi-Head Attention Block (PMA). A learnable set of seed tensors $\mathbf{S}_k$ is randomly initialised and used as the query for a cross attention operation with the inputs. The result is further processed by Self Attention Blocks. **Panel C:** The Self Attention Block (SAB), illustrated here with a pre-layer-normalization architecture. The input is normalised, processed by self attention, then further normalised and processed by an MLP. The block uses residual connections. **Panel D:** Illustration of the Masked (Self) Attention Block (MAB). The only difference compared to SAB is the edge mask, which follows the computation outlined in Panel A. **Panel E:** An instantiation of the ESA model. Edge inputs are processed by various different attention blocks, here in the order MSMSP, corresponding to masked, self, masked, self, and pooling attention layers.

3.1 Masked Attention Modules

We first introduce the basic notation and then give a formal description of the masked attention mechanism with a focus on the connectivity pattern specific to graphs. Following this, we provide algorithms for an efficient implementation of the masking operators using native tensor operations.

As in most graph learning settings, a graph is a tuple $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ represents the set of nodes (vertices), $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is the set of edges, and $N_n = |\mathcal{N}|$, $N_e = |\mathcal{E}|$. Nodes are associated with feature vectors $\mathbf{n}_i$ of dimension d_n for all nodes $i \in \mathcal{N}$, and d_e-dimensional edge features $\mathbf{e}_{ij}$ for all edges $e \in \mathcal{E}$. The node features are collected as rows in a matrix $\mathbf{N} \in \mathbb{R}^{N_n \times d_n}$, and similarly for edge features into $\mathbf{E} \in \mathbb{R}^{N_e \times d_e}$. The graph

Algorithm 1: Edge masking in PyTorch Geometric ($\mathbf{T}$ is the transpose; helper functions are explained in SI 6).

```
1   from esa import consecutive, first_unique_index
2   function edge_adjacency(batched_edge_index)
3       N_e ← batched_edge_index.size(1)
4       source_nodes ← batched_edge_index[0]
5       target_nodes ← batched_edge_index[1]
6
7       # unsqueeze and expand
8       exp_src ← source_nodes.unsq(1).expand((-1, N_e))
9       exp_trg ← target_nodes.unsq(1).expand((-1, N_e))
10
11      src_adj ←  exp_src == T(exp_src)
12      trg_adj ←  exp_trg == T(exp_trg)
13        cross ← (exp_src == T(exp_trg)) logical_or
14                (exp_trg == T(exp_src))
15
16      return (src_adj logical_or trg_adj logical_or cross)

17  function edge_mask(b_ei, b_map, B, L)
18      mask ← torch.full(size=(B, L, L), fill=False)
19      edge_to_graph ← b_map.index_select(0, b_ei[0, :])
20
21      edge_adj ← edge_adjacency(b_ei)
22      ei_to_original ← consecutive(
23          first_unique_index(edge_to_graph), b_ei.size(1))
24
25      edges ← edge_adj.nonzero()
26      graph_index ← edge_to_graph.idx_select(0, edges[:, 0])
27      coord_1 ← ei_to_original.idx_select(0, edges[:, 0])
28      coord_2 ← ei_to_original.idx_select(0, edges[:, 1])
29
30      mask[graph_index, coord_1, coord_2] ← True
31      return ~mask
```

connectivity information can be represented as an adjacency matrix $\mathbf{A}$, where $\mathbf{A}_{ij} = 1$ if $(i, j) \in \mathcal{E}$ and $\mathbf{A}_{ij} = 0$ otherwise. The edge list (edge index) representation is equivalent but more common in practice.

The main building block in ESA is masked scaled dot product attention. Panel A in Figure 3 illustrates this attention mechanism schematically. More formally, this attention mechanism is given by

$$\mathrm{SDPA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{M}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V} \tag{1}$$

where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are projections of the edge representations to queries, keys, and values, respectively. This is a minor extension of the standard scaled dot product attention via the additive mask $\mathbf{M}$ that can censor any of the pairwise attention scores, and d_k is the key dimension. The generalisation to masked multihead attention follows the same steps as in the original transformer [22]. Below and in the specification of the overall architecture, we refer to this function as MultiHead($\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{M}$).

Masked self-attention for graphs can be seen as graph-structured attention, and an instance of a generalized attention mechanism with custom attention patterns [28]. More specifically, in ESA the attention pattern is given by an *edge adjacency matrix* rather than allowing for interactions between all set items. Crucially, the edge adjacency matrix can be efficiently computed both for a single graph and for batched graphs using exclusively tensor operations. The case for a single graph is covered in the left side of Algorithm 1 through the `edge_adjacency` function. The first 3 lines of the function correspond to getting the number of edges, then separating the *source* and *target* nodes from the edge adjacency list (also called edge index), which is equivalent to the standard adjacency matrix of a graph. The source and target node tensors each have a dimension equal to the number of edges (N_e). Lines 7-9 add an additional dimension and efficiently repeat ('expand') the existing tensors to shape (N_e, N_e) without allocating new memory. Using the transpose, line 11 checks if the source nodes of any two edges are the same, and the same for target nodes on line 12. On line 13, cross connections are checked, where the source node of an edge is the target node of another, and vice versa. The operations of lines 11-14 result in boolean matrices of shape (N_e, N_e) which are summed for the final returned edge adjacency matrix. The right panel of Algorithm 1 depicts the case where the input graph represents a batch of smaller graphs. This requires an additional batch mapping tensor that maps each node in the batched graph to its original graph, and carefully manipulating the indices to create the final edge adjacency mask of shape (B, L, L), where L is the maximum number of edges in the batch.

Since in ESA attention is computed over edges, we chose to separate source and target node features for each edge, similarly to lines 4-5 of Algorithm 1, and concatenate them to the edge features:

$$\mathbf{x}_{ij} = \mathbf{n}_i \parallel \mathbf{n}_j \parallel \mathbf{e}_{ij} \tag{2}$$

for each edge e_{ij}. The resulting features $\mathbf{x}_{ij}$ are collected in a matrix $\mathbf{X} \in \mathbb{R}^{N_e \times (2d_n + d_e)}$.

Having defined the mask generation process and the masked multihead attention function, we next define the modular blocks of ESA, starting with the Masked Self Attention Block (MAB):

$$\mathrm{MAB}(\mathbf{X}, \mathbf{M}) = \mathbf{H} + \mathrm{MLP}(\mathrm{LayerNorm}(\mathbf{H})) \tag{3}$$

$$\mathbf{H} = \overline{\mathbf{X}} + \mathrm{MultiHead}(\overline{\mathbf{X}}, \overline{\mathbf{X}}, \overline{\mathbf{X}}, \mathbf{M}) \tag{4}$$

$$\overline{\mathbf{X}} = \mathrm{LayerNorm}(\mathbf{X}) \tag{5}$$

where the mask $\mathbf{M}$ is computed as in Algorithm 1 and has shape $B \times L \times L$, with B the batch size, and MLP is a multi-layer perceptron. A Self Attention Block (SAB) can be formally defined as:

$$\mathrm{SAB}(\mathbf{X}) = \mathrm{MAB}(\mathbf{X}, \mathbf{0}) \tag{6}$$

In principle, the masks can be arbitrary and a promising avenue of research could be in designing new mask

---

Algorithm 2: Node masking in the PyTorch Geometric framework.

```
1  from torch_geometric import unbatch_edge_index
2  function node_mask(batched_edge_index, batch_map, B, M)
3      # batched_edge_index batches all edge_index tensors into a single tensor
4      # batch_map maps nodes to graphs
5      # B, M are the batch, respectively mask size
6      mask ← torch.full(size=(B, M, M), fill=False)
7      graph_idx ← batch_map.index_select(0, batched_edge_index[0, :])
8      edge_index_list ← unbatch_edge_index(batched_edge_index, batch_map)
9      edge_index ← torch.cat(edge_index_list, dim=1)
10     mask[graph_idx, edge_index[0, :], edge_index[1, :]] ← True
11     return ~mask
```

---

types. For certain tasks such as node classification, we also defined an alternative version of ESA for nodes (NSA). The only major difference is the mask generation step. The single graph case is trivial as all the masking information is available in the edge index, so we provide the general batched case in Algorithm 2.

3.2   ESA Architecture

Our architecture consists of two components: *i*) an encoder that interleaves masked and self-attention blocks to learn an effective representation of edges, and *ii*) a pooling block based on multi-head attention, inspired by the decoder component from the set transformer architecture [39]. The latter comes naturally when one considers graphs as sets of edges (or nodes), as done here. ESA is a purely attention-based architecture that leverages the scaled dot product mechanism proposed by Vaswani et al. [22] through masked- and standard self-attention blocks (MABs and SABs) and a pooling by multi-head attention (PMA) block.

The encoder consisting of arbitrarily interleaved MAB and SAB blocks is given by:

$$\mathrm{Encoder}(\mathbf{X}, \mathbf{M}) = \mathrm{AB} \circ \mathrm{AB} \circ \cdots \circ \mathrm{AB}(\mathbf{X}, \mathbf{M}), \quad \text{where} \quad \mathrm{AB} \in \{\mathrm{MAB}, \mathrm{SAB}\} \tag{7}$$

Here, 'AB' refers to an *attention block* that can be instantiated as an MAB or SAB.

The pooling module that is responsible for aggregating the processed edge representations into a graph-level representation is formally defined by:

$$\mathrm{PMA}_{k,p}(\mathbf{Z}) = \mathrm{SAB}^p \left(\overline{\mathbf{S}} + \mathrm{MLP}(\overline{\mathbf{S}}) \right) \tag{8}$$

$$\overline{\mathbf{S}} = \mathrm{LayerNorm} \left(\mathrm{MultiHead} \left(\mathbf{S}_k, \mathbf{Z}, \mathbf{Z}, \mathbf{0} \right) \right) \tag{9}$$

where $\mathbf{S}_k$ is a tensor of k learnable seed vectors that are randomly initialised and $\mathrm{SAB}^p(\cdot)$ is the application of p SABs. Technically, it suffices to set $k = 1$ to output a single representation for the entire graph. However, we have empirically found it beneficial to set it to a small value, such as $k = 32$. Moreover, this change allows self attention (SABs) to further process the k resulting representations, which can be simply summed or averaged due to the small k. Contrary to classical readouts that aggregate directly over set items (i.e., nodes), pooling by multi-head attention performs the final aggregation over the embeddings of learnt seed vectors S_k. While tasks involving node-level predictions require only the Encoder component, the predictive tasks involving graph-level representations require all the modules, both the encoder and pooling by multi-head attention. The architecture in the latter setting is formally given by

$$\mathbf{Z}_{\mathrm{out}} = \mathrm{PMA}_{k,p}(\mathrm{Encoder}(\mathbf{X}, \mathbf{M}) + \mathbf{X})$$

As optimal configurations are task specific, we do not explicitly fix all the architectural details. For example, it is possible to select between layer and batch normalisation, a pre-LN or post-LN architecture [40], or standard and gated MLPs, along with GLU variants, e.g., SwiGLU [41].

3.3   Time and Memory Scaling

ESA is enabled by recent advances in efficient and exact attention, such as memory-efficient attention and Flash attention [42–45]. Flash attention does not yet support masking, but memory-efficient implementations with arbitrary masking capabilities exist in both PyTorch and xFormers [46, 47]. Theoretically, the memory complexity of memory-efficient approaches is $O(\sqrt{n})$, where n is the sequence/set size, with the same time complexity as standard attention. Since all operations in ESA except the cross-attention in PMA are based on self-attention, our method benefits directly from all of these advances. Moreover, our cross-attention is performed between the full set of size n and a small set of $k \leq 32$ learnable seeds, such that the complexity of the operation is $O(kn)$ even for standard attention. Flash attention, which has linear memory complexity even for self-attention and is up to ×25 times faster than standard attention, supports cross-attention and we use it for further uplifts. However, these implementations are not optimised for graph learning and we have found that the biggest bottleneck is not the attention computation, but rather storing the dense edge adjacency mask, which requires memory quadratic in the number of edges. An evident, but not yet available, optimisation could amount to storing the masks in a sparse tensor format. Alternatively, even a single boolean requires an

Table 1: The table reports root mean squared error on QM9 (RMSE is the standard metric for quantum mechanics) and R^2 for DOCKSTRING (DOCK) and MOLECULENET (MN), presented as mean $\pm$ standard deviation over 5 runs. The mean absolute error (MAE) is reported for PCQM4MV2 (MAE is the standard metric for this task) over a single run due to the size of the dataset. The lowest MAEs and RMSEs, and the highest R^2 values are highlighted in bold. OOM denotes out-of-memory errors. A complete table, including GCN and GIN, is provided in Supplementary Table 1.

	Target	DropGIN	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	ESA
QM9 ($\downarrow$)	μ	0.55 ± 0.01	0.55 ± 0.01	0.55 ± 0.01	$\mathbf{0.53 \pm 0.01}$	0.63 ± 0.01	0.76 ± 0.02	0.94 ± 0.17	0.56 ± 0.00
	α	0.44 ± 0.03	0.48 ± 0.02	0.46 ± 0.02	0.48 ± 0.07	0.40 ± 0.02	0.45 ± 0.01	0.60 ± 0.13	$\mathbf{0.40 \pm 0.00}$
	ϵ_{HOMO}	0.11 ± 0.00	0.11 ± 0.00	0.10 ± 0.00	$\mathbf{0.10 \pm 0.00}$	0.11 ± 0.00	0.12 ± 0.00	0.11 ± 0.00	0.10 ± 0.00
	ϵ_{LUMO}	0.12 ± 0.00	0.12 ± 0.00	0.13 ± 0.00	$\mathbf{0.11 \pm 0.00}$	0.11 ± 0.00	0.13 ± 0.00	0.11 ± 0.00	0.11 ± 0.00
	$\Delta\epsilon$	0.17 ± 0.00	0.16 ± 0.01	0.16 ± 0.00	$\mathbf{0.14 \pm 0.00}$	0.16 ± 0.00	0.18 ± 0.01	0.15 ± 0.01	0.15 ± 0.00
	$\langle R^2 \rangle$	29.87 ± 0.34	30.91 ± 0.15	30.15 ± 0.36	28.50 ± 0.44	29.63 ± 0.35	31.54 ± 0.42	30.42 ± 1.19	$\mathbf{28.33 \pm 0.32}$
	ZPVE	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.03 ± 0.00	0.06 ± 0.03	0.03 ± 0.00	0.03 ± 0.01	$\mathbf{0.03 \pm 0.00}$
	U_0	29.82 ± 6.32	27.82 ± 3.98	25.40 ± 3.62	31.64 ± 6.49	24.60 ± 4.70	15.48 ± 4.67	14.50 ± 4.34	$\mathbf{4.78 \pm 0.67}$
	U	24.74 ± 6.31	30.91 ± 2.62	24.92 ± 4.18	25.04 ± 5.48	15.55 ± 8.23	12.45 ± 1.86	15.82 ± 4.26	$\mathbf{6.80 \pm 2.81}$
	H	34.02 ± 5.66	28.92 ± 4.11	23.42 ± 2.77	27.34 ± 8.17	19.01 ± 4.65	11.42 ± 3.72	12.92 ± 4.29	$\mathbf{6.02 \pm 1.32}$
	G	25.41 ± 4.86	31.49 ± 1.86	23.24 ± 2.54	22.31 ± 3.45	31.20 ± 4.61	29.72 ± 8.79	13.08 ± 4.81	$\mathbf{6.10 \pm 1.33}$
	c_{V}	0.19 ± 0.01	0.19 ± 0.01	0.19 ± 0.00	0.18 ± 0.01	0.18 ± 0.02	0.18 ± 0.00	0.19 ± 0.03	$\mathbf{0.16 \pm 0.00}$
	U_0^{ATOM}	0.38 ± 0.02	0.39 ± 0.03	0.38 ± 0.02	0.35 ± 0.02	0.29 ± 0.00	0.30 ± 0.02	0.33 ± 0.05	$\mathbf{0.24 \pm 0.01}$
	U^{ATOM}	0.40 ± 0.04	0.48 ± 0.07	0.40 ± 0.04	0.36 ± 0.02	0.30 ± 0.01	0.30 ± 0.01	0.33 ± 0.06	$\mathbf{0.24 \pm 0.01}$
	H^{ATOM}	0.38 ± 0.04	0.38 ± 0.02	0.41 ± 0.06	0.37 ± 0.03	0.30 ± 0.01	0.31 ± 0.01	0.36 ± 0.09	$\mathbf{0.25 \pm 0.00}$
	G^{ATOM}	0.37 ± 0.04	0.34 ± 0.02	0.33 ± 0.01	0.31 ± 0.02	0.26 ± 0.01	0.27 ± 0.01	0.36 ± 0.08	$\mathbf{0.22 \pm 0.02}$
	A	0.90 ± 0.06	0.97 ± 0.12	1.08 ± 0.16	1.01 ± 0.10	64.88 ± 29.77	3.82 ± 2.05	1.42 ± 0.44	$\mathbf{0.75 \pm 0.11}$
	B	0.19 ± 0.05	0.21 ± 0.04	0.26 ± 0.01	0.26 ± 0.02	0.10 ± 0.03	0.11 ± 0.01	0.16 ± 0.05	$\mathbf{0.08 \pm 0.01}$
	C	0.27 ± 0.02	0.27 ± 0.01	0.28 ± 0.00	0.28 ± 0.01	0.12 ± 0.04	0.10 ± 0.02	0.12 ± 0.05	$\mathbf{0.05 \pm 0.01}$
DOCK ($\uparrow$)	ESR2	0.68 ± 0.00	0.67 ± 0.00	0.65 ± 0.00	0.70 ± 0.00	OOM	0.64 ± 0.01	0.68 ± 0.00	$\mathbf{0.70 \pm 0.00}$
	F2	0.89 ± 0.00	0.89 ± 0.00	0.89 ± 0.00	0.89 ± 0.00	OOM	0.87 ± 0.01	0.88 ± 0.00	$\mathbf{0.89 \pm 0.00}$
	KIT	0.83 ± 0.00	0.83 ± 0.00	0.83 ± 0.00	$\mathbf{0.84 \pm 0.00}$	OOM	0.80 ± 0.01	0.83 ± 0.00	0.84 ± 0.00
	PARP1	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	0.92 ± 0.00	OOM	0.91 ± 0.00	0.92 ± 0.01	$\mathbf{0.93 \pm 0.00}$
	PGR	0.70 ± 0.00	0.68 ± 0.01	0.67 ± 0.01	0.72 ± 0.00	OOM	0.68 ± 0.01	0.70 ± 0.01	$\mathbf{0.73 \pm 0.00}$
MN ($\uparrow$)	FREESOLV	0.97 ± 0.00	0.96 ± 0.01	0.97 ± 0.01	0.95 ± 0.01	0.93 ± 0.00	0.93 ± 0.02	0.86 ± 0.03	$\mathbf{0.98 \pm 0.00}$
	LIPO	0.81 ± 0.01	0.82 ± 0.01	0.82 ± 0.01	$\mathbf{0.83 \pm 0.01}$	0.61 ± 0.04	0.55 ± 0.02	0.79 ± 0.00	0.81 ± 0.01
	ESOL	0.94 ± 0.01	0.93 ± 0.01	0.93 ± 0.00	0.94 ± 0.01	0.91 ± 0.02	0.89 ± 0.03	0.91 ± 0.00	$\mathbf{0.94 \pm 0.00}$
PCQM4MV2 ($\downarrow$)		N/A	N/A	N/A	N/A	N/A	N/A	N/A	$\mathbf{0.0235}$

Table 2: The table reports mean absolute error (MAE) on the ZINC dataset, presented as mean $\pm$ standard deviation over 5 runs. The best/lowest value is highlighted in bold. A complete table, including GCN, GIN, and DropGIN is provided in Supplementary Table 2.

Dataset ($\downarrow$)	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	ESA	ESA (PE)
ZINC	0.078 ± 0.01	0.079 ± 0.00	0.057 ± 0.01	0.036 ± 0.00	0.047 ± 0.01	0.024 ± 0.01	0.027 ± 0.00	$\mathbf{0.017 \pm 0.00}$

entire byte of storage in PyTorch, increasing the theoretical memory usage by 8 times. Further limitations and possible optimisations are discussed in SI 5. Despite some of these limitations, we have successfully trained ESA models for graphs with up to approximately 30,000 edges (e.g., DD in Table 6).

4 Results

We perform a comprehensive evaluation of ESA on 70 different tasks, including domains such as molecular property prediction, vision graphs, and social networks, as well as different aspects of representation learning on graphs, ranging from node-level tasks with homophily and heterophily graph types to modelling long range dependencies, shortest paths, and 3D atomic systems. We quantify the performance of our approach relative to 6 GNN baselines: GCN, GAT, GATv2, PNA, GIN, and DropGIN (more expressive than 1-WL), and 3 graph transformer baselines: Graphormer, TokenGT, and GraphGPS. All the details on hyper-parameter tuning, rationale for the selected metrics, and selection of baselines can be found in SI 7.1 to SI 7.3. In the remainder of the section, we summarize our findings across molecular learning, mixed graph-level tasks, node-level tasks, and ablations on the interleaving operator along with insights on time and memory scaling.

4.1 Molecular Learning

As learning on molecules has emerged as one of the most successful applications of graph learning, we present an in-depth evaluation including quantum mechanics, molecular docking, and various physical chemistry and biophysics benchmarks, as well as an exploration of learning on 3D atomic systems, transfer learning, and learning on large molecules of therapeutic relevance (peptides).

QM9. We report results for all 19 QM9 [48] targets in Table 1, with GCN and GIN separately in Supplementary Table 1 due to space restrictions. We observe that on 15 out of 19 properties, ESA is the best performing model. The exceptions are the frontier orbital energies (HOMO and LUMO energy, HOMO-LUMO gap) and the dipole moment (μ), where PNA is slightly ahead of it. Other graph transformers are competitive relative to GNNs on many properties but vary in performance across tasks.

DOCKSTRING. DOCKSTRING [49] is a recent drug discovery data collection consisting of molecular docking scores for 260,155 small molecules and 5 high-quality targets from different protein families that were selected

Table 3: The table reports Matthews correlation coefficient (MCC) for graph-level molecular classification tasks from MoleculeNet and National Cancer Institute (NCI), presented as mean ± standard deviation over 5 different runs. OOM denotes out-of-memory errors. The highest mean values are highlighted in bold. A complete table, including GCN and GIN, is provided in Supplementary Table 4 with MCC as metric, and in Supplementary Table 5 with accuracy.

	Data (↑)	DropGIN	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	ESA
MN	BBBP	0.68 ± 0.02	0.74 ± 0.01	0.73 ± 0.03	0.73 ± 0.03	0.55 ± 0.01	0.58 ± 0.07	0.70 ± 0.04	**0.84 ± 0.01**
	BACE	0.65 ± 0.03	0.63 ± 0.02	0.64 ± 0.03	0.64 ± 0.02	0.52 ± 0.02	0.58 ± 0.03	0.62 ± 0.03	**0.72 ± 0.02**
	HIV	0.46 ± 0.03	0.42 ± 0.06	0.34 ± 0.06	0.42 ± 0.04	OOM	0.46 ± 0.02	0.25 ± 0.21	**0.53 ± 0.01**
NCI	NCI1	0.69 ± 0.03	0.70 ± 0.02	0.65 ± 0.03	0.70 ± 0.03	0.54 ± 0.02	0.53 ± 0.03	0.70 ± 0.03	**0.75 ± 0.01**
	NCI109	0.68 ± 0.02	0.66 ± 0.01	0.66 ± 0.01	0.67 ± 0.02	0.50 ± 0.02	0.45 ± 0.03	0.62 ± 0.01	**0.70 ± 0.01**

Table 4: The table reports mean absolute error (MAE) and average precision (AP) for two long-range molecular benchmarks involving peptides. The molecular graph of a peptide is much larger than that of a small drug-like molecule and this makes the tasks well-suited for long-range benchmarking. All the results except the ones for ESA and TokenGT are extracted from [29]. The number of layers for PEPT-STRUCT, respectively PEPT-FUNC is given as $(\cdot/\cdot)$.

Dataset	GCN (6/6)	GIN (10/8)	GPS (8/6)	TokenGT (10/10)	ESA (3/4)
PEPT-STR (MAE ↓)	0.2460 ± 0.0007	0.2473 ± 0.0017	0.2509 ± 0.0014	0.2489 ± 0.0013	**0.2453 ± 0.0003**
PEPT-FN (AP ↑)	0.6860 ± 0.0050	0.6621 ± 0.0067	0.6534 ± 0.0091	0.6263 ± 0.0117	**0.6863 ± 0.0044**

as a regression benchmark, with different levels of difficulty: PARP1 (enzyme, easy), F2 (protease, easy to medium), KIT (kinase, medium), ESR2 (nuclear receptor, hard), and PGR (nuclear receptor, hard). We report results for the 5 targets in Table 1 (and Supplementary Table 1) and observe that ESA is the best performing method on 4 out of 5 tasks, with PNA slightly ahead on the medium-difficulty KIT. TokenGT and GraphGPS do not generally match ESA or even PNA. Molecular docking scores also depend heavily on the 3D geometry, as discussed in the original paper [49], posing a difficult challenge for all methods. Interestingly, not only ESA but all tuned GNN baselines outperform the strongest method in the original manuscript (Attentive FP, a GNN based on attention [50]) despite using 20,000 less training molecules (which we leave out as a validation set). This illustrates the importance of evaluation relative to baselines with tuned hyper-parameters.

MoleculeNet and NCI. We report results for a varied selection of three regression and three classification benchmarks from MoleculeNet, as well as two benchmarks from the National Cancer Institute (NCI) consisting of compounds screened for anti-cancer activity (Tables 1 and 3, and Supplementary Tables 1 and 4). We also report the accuracy in Supplementary Table 5. Except HIV, these datasets pose a challenge to graph transformers due to their small size (<5,000 compounds). With the exception of the Lipophilicity (LIPO) dataset from MoleculeNet, we observe that ESA is the preferred method, and often by a significant margin, for example on BBBP and BACE, despite their small size (2,039, respectively 1,513 total compounds before splitting). On the other hand, Graphormer and TokenGT perform poorly, possibly due to the small-sample nature of these tasks. GraphGPS is closer to the top performers but still underperforms compared to GAT(v2) and PNA. These results also show the importance of appropriate evaluation metrics, as the accuracy on HIV for all methods is above 97% (Supplementary Table 5), but the MCC (Table 3) is significantly lower.

PCQM4MV2. PCQM4MV2 is a quantum chemistry benchmark introduced as a competition through the Open Graph Benchmark Large-Scale Challenge (OGB-LSC) project [51]. It consists of 3,378,606 training molecules with the goal of predicting the DFT-calculated HOMO-LUMO energy gap from 2D molecular graphs. It has been widely used in the literature, especially to evaluate graph transformers [25, 26, 52]. Since the test splits are not public, we use the available validation set (73,545 molecules) as a test set, as is often done in the literature, and report results on it after training for 400 epochs. We report results from a single run, as it is common in the field due to the large size of the dataset [25, 26, 52]. For the same reason, we do not run our baselines and instead chose to focus on the state-of-the-art results published on the official leaderboard[*]. At the time of writing (November 2024), the best performing model achieves a validation set MAE of 0.0671 [53]. Here, ESA achieves a MAE of **0.0235**, which is almost 3 times lower. It it worth noting that the top 3 methods for this dataset are all bespoke architectures designed for molecular learning, for example Uni-Mol+ [54], Transformer-M [55], and TGT [53]. In contrast, ESA is general purpose (does not use any information or technique specifically for molecular learning), uses only the 2D input graph, and does not use any positional or structural encodings.

ZINC. We report results on the full ZINC dataset with 250,000 compounds (Table 2), which is commonly used for generative purposes [56, 57]. This is one of the only benchmarks where the graph transformer baselines (Graphormer, TokenGT, GraphGPS) convincingly outperformed strong GNN baselines. ESA, without positional encodings, slightly underperforms compared to GraphGPS, which uses random-walk structural encodings (RWSE). This type of encoding is known to be beneficial in molecular tasks, and especially for ZINC [27, 58]. Thus, we also evaluated an ESA + RWSE model, which increased relative performance by almost 40%. While there is no leaderboard available for the full version of ZINC, recently Ma et al. [31] evaluated their own method (GRIT) against 11 other baselines, including higher-order GNNs, with

---

[*]https://ogb.stanford.edu/docs/lsc/leaderboards/#pcqm4mv2

Table 5: A summary of the transfer learning performance on QM9 for HOMO and LUMO properties, presented as mean ± standard deviation over 5 different runs. The metric is the root mean squared error (RMSE). All the models use the 3D atomic coordinates and atom types as inputs and no other node or edge features. 'Strat.' stands for strategy and specifies the type of learning: GW only (no transfer learning), inductive, or transductive. The lowest values are highlighted in bold. A complete table, including GCN and GIN, is provided in Supplementary Table 3.

Task	Strat.	DropGIN	GAT	GATv2	PNA	Grph.	TokenGT	GPS	ESA
HOMO	GW	0.162 ± 0.00	0.159 ± 0.00	0.157 ± 0.00	$\mathbf{0.151 \pm 0.00}$	0.179 ± 0.01	0.200 ± 0.01	0.162 ± 0.00	0.152 ± 0.00
	Ind.	0.136 ± 0.00	0.131 ± 0.00	0.133 ± 0.00	0.132 ± 0.00	0.134 ± 0.00	0.156 ± 0.00	0.151 ± 0.00	$\mathbf{0.131 \pm 0.00}$
	Trans.	0.126 ± 0.00	0.123 ± 0.00	0.124 ± 0.00	0.121 ± 0.00	0.125 ± 0.00	0.137 ± 0.00	0.147 ± 0.00	$\mathbf{0.119 \pm 0.00}$
LUMO	GW	0.180 ± 0.00	0.181 ± 0.00	0.178 ± 0.00	0.174 ± 0.00	0.190 ± 0.01	0.204 ± 0.01	0.178 ± 0.00	$\mathbf{0.174 \pm 0.00}$
	Ind.	0.159 ± 0.00	0.156 ± 0.00	0.157 ± 0.00	0.156 ± 0.00	0.151 ± 0.00	0.165 ± 0.00	0.167 ± 0.00	$\mathbf{0.150 \pm 0.00}$
	Trans.	0.157 ± 0.00	0.153 ± 0.00	0.153 ± 0.00	0.153 ± 0.00	0.147 ± 0.00	0.156 ± 0.00	0.169 ± 0.00	$\mathbf{0.146 \pm 0.00}$

the best reported MAE being $\mathbf{0.023 \pm 0.001}$. This shows that ESA can already almost match state-of-the-art models without structural encodings, and significantly improves upon this result when augmented.

Long-range peptide tasks. Graph learning with transformers has traditionally been evaluated on long-range graph benchmarks (LRGB) [59]. However, it was recently shown that simple graph neural networks outperform most attention-based methods [29]. We selected two long-range benchmarks involving peptide property prediction: PEPTIDES-STRUCT and PEPTIDES-FUNC. From the LRGB collection, these two stand out due to having the longest average shortest path (20.89 ± 9.79 versus 10.74 ± 0.51 for the next) and the largest average diameter (56.99 ± 28.72 versus 27.62 ± 2.13). We report ESA results against tuned GNNs and GraphGPS models from [59], as well as our own optimised TokenGT model (Table 4). Despite using only half the number of layers as other methods or less, ESA outperformed these baselines and matched the second model on the PEPT-STRUCT leaderboard, and is within the top 5 for PEPT-FUNC (as of November 2024).

Learning on 3D atomic systems. We adapt a subset from the Open Catalyst Project (OCP) [60, 61] for evaluation, with the full steps in SI 9, including deriving edges from atom positions based on a cutoff, pre-processing specific to crystal systems, and encoding atomic distances using Gaussian basis functions. As a prototype, we compare NSA (MAE of $\mathbf{0.799 \pm 0.008}$) against Graphormer (0.839 ± 0.005), one of the best models that have been used for the Open Catalyst Project [62]. These encouraging results on a subset of OCP motivated us to further study modelling 3D atomic system through the lens of transfer learning.

Transfer learning on frontier orbital energies. We follow the recipe recently outlined for drug discovery and quantum mechanics by [32] and leverage a recent, refined version of the QM9 HOMO and LUMO energies [63] that provides alternative DFT calculations and new calculations at the more accurate GW level of theory. As outlined by [32], transfer learning can occur *transductively* or *inductively*. The transfer learning scenario is important and is detailed in SI 4. We use a 25K/5K/10K train/validation/test split for the high-fidelity GW data, and we train separate low-fidelity DFT models on the entire dataset (transductive) or with the high-fidelity test set molecules removed (inductive). Since the HOMO and LUMO energies depend to a large extent on the molecular geometry, we use the 3D-aware version of ESA from the previous section, and adapt all of our baselines to the 3D setup. Our results are reported in Table 5. Without transfer learning (strategy 'GW' in Table 5), ESA and PNA are almost evenly matched, which is already an improvement since PNA was better for frontier orbital energies without 3D structures (Table 1), while the graph transformers perform poorly. Employing transfer learning all the methods improve significantly, but ESA outperforms all baselines for both HOMO and LUMO, in both transductive and inductive tasks.

4.2 Mixed Graph-level Tasks

Other than molecular learning, we examine a suite of graph-level benchmarks from various domains, including computer vision, bioinformatics, synthetic graphs, and social graphs. Our results are reported in Table 6 and Supplementary Table 7, where ESA generally outperforms all the baselines. In the context of state-of-the-art models from online leaderboards, of particular note are the accuracy results (provided in Supplementary Table 7) on the vision datasets, where on MNIST ESA matches the best performing model [64], and is in the top 5 on CIFAR10 at the time of submission (November 2024), without positional or structural encodings or other helper mechanisms. Similarly, on the MALNETTINY dataset of function call graphs popularised by GraphGPS [27], we achieve the highest recorded accuracy.

4.3 Node-level Benchmarks

Node-level tasks are an interesting challenge for our proposed approach as the PMA module is not needed and node-level propagation is the most natural strategy of learning node representations. To this end, we did prototype with an edge-to-node pooling module which would allow ESA to learn node embeddings, with good results. However, the approach does not currently scale to the millions of edges that some heterophilous graphs have. For these reasons, we revert to the simpler node-set attention (NSA) formulation. Table 7 summarizes our results, indicating that NSA performs well on all 11 node-level benchmarks, including homophilous (citation), heterophilous, and shortest path tasks. We have adapted Graphormer and TokenGT for node

Table 6: The table reports Matthews correlation coefficient (MCC) for graph-level classification tasks from various domains, presented as mean ± standard deviation over 5 different runs. OOM denotes out-of-memory errors, and N/A that the model is unavailable (e.g., node/edge features are not integers, which are required for Graphormer and TokenGT). The poor performance of GraphGPS on MALNETTINY in Table 6 can be explained by our use of one-hot degrees as node features for datasets lacking pre-computed features, while GraphGPS originally used the more informative local degree profile [65]. The highest mean values are highlighted in bold. A complete table, including GCN and GIN, is provided in Supplementary Table 6 with MCC as metric, and in Supplementary Table 7 with accuracy.

	Data	DropGIN	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	ESA
	MALNETTINY	0.90 ± 0.01	0.90 ± 0.01	0.90 ± 0.01	0.91 ± 0.01	OOM	0.78 ± 0.01	0.79 ± 0.01	**0.93 ± 0.00**
VIS.	MNIST	0.97 ± 0.00	0.97 ± 0.00	0.98 ± 0.00	0.98 ± 0.00	N/A	N/A	0.98 ± 0.00	**0.99 ± 0.00**
	CIFAR10	0.61 ± 0.01	0.66 ± 0.01	0.66 ± 0.01	0.69 ± 0.01	N/A	N/A	0.71 ± 0.01	**0.73 ± 0.00**
BIO.	ENZYMES	0.58 ± 0.04	0.75 ± 0.02	0.74 ± 0.02	0.68 ± 0.03	N/A	N/A	0.73 ± 0.05	**0.75 ± 0.01**
	PROTEINS	0.46 ± 0.01	0.46 ± 0.04	0.49 ± 0.04	0.47 ± 0.07	N/A	N/A	0.44 ± 0.02	**0.59 ± 0.02**
	DD	0.54 ± 0.07	0.47 ± 0.05	0.53 ± 0.03	0.56 ± 0.08	OOM	0.46 ± 0.05	0.60 ± 0.05	**0.65 ± 0.03**
SYNTH	SYNTH	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	N/A	N/A	1.00 ± 0.00	**1.00 ± 1.00**
	SYNT N.	0.97 ± 0.05	0.76 ± 0.07	0.91 ± 0.07	1.00 ± 0.00	N/A	N/A	1.00 ± 0.00	**1.00 ± 0.00**
	SYNTHIE	0.94 ± 0.02	0.70 ± 0.04	0.80 ± 0.04	0.88 ± 0.06	N/A	N/A	**0.95 ± 0.02**	0.95 ± 0.02
SOCIAL	IMDB-B	0.61 ± 0.06	0.69 ± 0.04	0.60 ± 0.05	0.56 ± 0.07	0.56 ± 0.05	0.61 ± 0.05	0.60 ± 0.05	**0.74 ± 0.03**
	IMDB-M	0.12 ± 0.10	0.20 ± 0.05	0.20 ± 0.03	0.03 ± 0.07	0.22 ± 0.03	0.20 ± 0.02	0.23 ± 0.02	**0.25 ± 0.03**
	TWITCH E.	0.38 ± 0.01	0.37 ± 0.01	0.37 ± 0.01	0.08 ± 0.16	0.39 ± 0.00	0.39 ± 0.00	0.40 ± 0.00	**0.40 ± 0.00**
	RDT. THR.	0.56 ± 0.01	0.53 ± 0.02	0.54 ± 0.02	0.11 ± 0.23	0.57 ± 0.00	0.56 ± 0.00	**0.57 ± 0.00**	0.57 ± 0.00

Table 7: The table reports Matthews correlation coefficient (MCC) for 11 node-level classification tasks, presented as mean ± standard deviation over 5 different runs. The number of nodes for the shortest path (SP) benchmarks is given in parentheses (based on randomly-generated 'infected' Erdős–Rényi (ER) graphs; details are provided in SI 8). For SQUIRREL and CHAMELEON, we used the filtered datasets introduced by Platonov et al. [66] to fix existing data leaks. The highest mean values are highlighted in bold. Additional heterophily results are provided in Supplementary Table 10. A complete table, including GIN and DropGIN, is provided in Supplementary Table 8 with MCC as the performance metric, and in Supplementary Table 9 with accuracy.

	Data (↑)	GCN	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	NSA
	PPI	0.98 ± 0.00	**0.99 ± 0.00**	0.98 ± 0.01	0.99 ± 0.00	N/A	N/A	N/A	0.99 ± 0.00
CITE	CITESEER	0.61 ± 0.01	0.59 ± 0.03	0.61 ± 0.01	0.51 ± 0.03	OOM	0.38 ± 0.02	0.54 ± 0.01	**0.63 ± 0.00**
	CORA	0.77 ± 0.01	0.75 ± 0.01	0.73 ± 0.01	0.64 ± 0.03	OOM	0.37 ± 0.18	0.64 ± 0.04	**0.77 ± 0.00**
HETERO	ROMAN E.	0.47 ± 0.00	0.74 ± 0.01	0.76 ± 0.00	0.86 ± 0.00	N/A	N/A	0.84 ± 0.01	**0.87 ± 0.00**
	AMAZON R.	0.18 ± 0.00	0.26 ± 0.01	0.25 ± 0.01	0.21 ± 0.02	N/A	N/A	0.11 ± 0.14	**0.34 ± 0.01**
	MINESWEEPER	0.30 ± 0.00	0.48 ± 0.02	0.51 ± 0.01	0.62 ± 0.04	N/A	N/A	0.56 ± 0.01	**0.69 ± 0.00**
	TOLOKERS	0.30 ± 0.01	0.38 ± 0.01	0.39 ± 0.00	0.35 ± 0.05	N/A	N/A	0.35 ± 0.02	**0.43 ± 0.00**
	SQUIRREL	0.20 ± 0.01	0.24 ± 0.02	0.24 ± 0.01	0.22 ± 0.01	0.10 ± 0.09	0.18 ± 0.02	0.23 ± 0.02	**0.29 ± 0.01**
	CHAMELEON	0.32 ± 0.01	0.24 ± 0.06	0.28 ± 0.03	0.27 ± 0.03	0.25 ± 0.04	0.26 ± 0.04	0.30 ± 0.08	**0.39 ± 0.02**
SP	ER (15K)	0.22 ± 0.02	0.32 ± 0.00	0.32 ± 0.00	0.54 ± 0.09	OOM	0.06 ± 0.00	0.18 ± 0.04	**0.92 ± 0.01**
	ER (30K)	0.09 ± 0.03	0.10 ± 0.06	0.10 ± 0.06	0.42 ± 0.05	OOM	OOM	OOM	**0.87 ± 0.01**

Table 8: Example of multiple ESA configurations and their impact on performance via three different kinds of benchmarks: a graph-level molecular task (BBBP), a graph-level vision task (MNIST), and a node-level heterophilous task (CHAMELEON). In the model configurations, 'M' denotes a MAB, 'S' a SAB before the PMA module and 'S' afterwards, and 'P' is the PMA module. The performance metric is the Matthews correlation coefficient (MCC).

Dataset	Model	MCC (↑)	Dataset	Model	MCC (↑)	Dataset	Model	MCC (↑)
BBBP	M M M M S P S	0.845	MNIST	S S M M M M M M M P S	0.986	CHAME-LEON	M S M S M S	0.422
	M M M S P S S	0.835		S M S M M M S M P	0.983		S S M M S S	0.384
	S S S M M P S	0.812		S S S M M M S S S P	0.982		S M S M S M	0.378
	M M M M M P	0.782		M S M S M S M S M S P	0.980		M M M M	0.359
	S M S M S P	0.768		M M M M M M M M M P	0.980		S S M M M M	0.351

classification as this functionality was not originally available, although they require integer node and edge features which restricts their use on some datasets (denoted by N/A in Table 7). NSA achieves the highest MCC score on homophilous and heterophilous graphs, but GCN has the highest accuracy on CORA (Supplementary Table 9). Some methods, for instance GraphSAGE [67] are designed to perform particularly well under heterophily and to account for that we include GraphSAGE [67] and Graph Transformer [33] as the two top performing baselines from Platonov et al. [66] in our extended results in Supplementary Table 10. With the exception of the AMAZON RATINGS datasets, we outperform the two baselines by a noticeable margin. Our results on CHAMELEON stand out in particular, as well as on the shortest path benchmarks, where other graph transformers are unable to learn and even PNA fails to be competitive.

4.4 Effects of Varying the Layer Order and Type

In Table 8, we summarise the results of an ablation relative to the interleaving operator with different order and types of layers in our architecture. Smaller datasets perform well with 4 to 6 feature extraction layers, while larger datasets with more complex graphs like MNIST benefit from up to 10 layers. We have generally observed that the top configurations tend to include self-attention layers at the front, with masked attention layers in the middle and self-attention layers at the end, surrounding the PMA readout. Naive configurations such as all-masked layers or simply alternating masked and self-attention layers do not tend to be optimal for
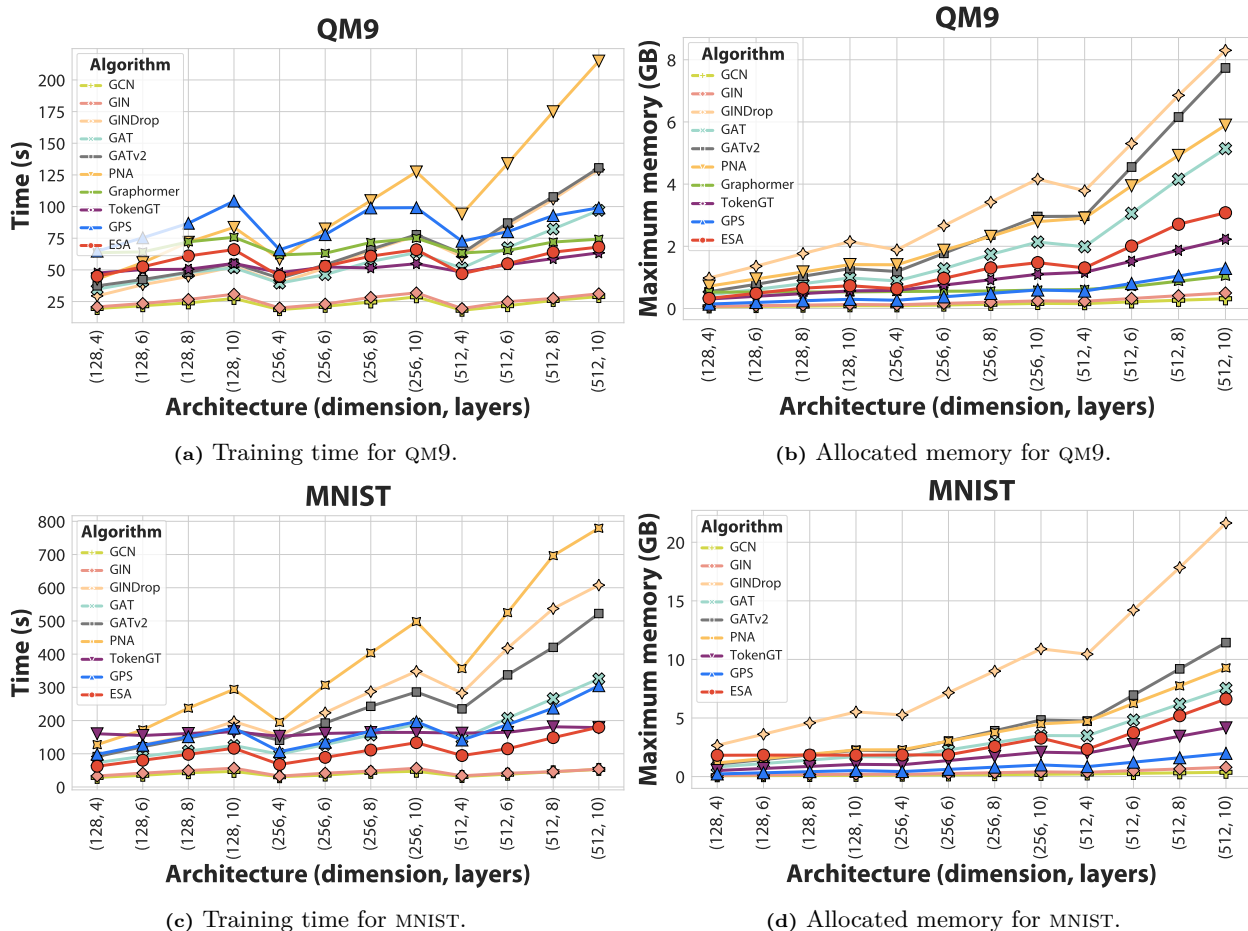
(a) Training time for QM9.

(b) Allocated memory for QM9.

(c) Training time for MNIST.

(d) Allocated memory for MNIST.

Figure 4: The elapsed time for training a single epoch for different datasets (in seconds), and the maximum allocated memory during this training epoch (GB). Different configurations are tested, while varying hidden dimensions and number of layers. QM9 has around 130K small molecules (a maximum of 29 nodes and 56 edges), DOCKSTRING has around 260K graphs that are around 6 times larger, and finally MNIST has 70K graphs which are around 11-12 times larger than QM9. We use dummy integer features for TokenGT when benchmarking MNIST. Graphormer runs out of memory for DOCKSTRING and MNIST.

graph-level prediction tasks. This ablation experiment demonstrates the importance of vertical combination of masked and self-attention layers for the performance of our model.

4.5 Time and Memory Scaling

The theoretical properties of ESA are discussed in *Methods*, Section 3.3, where we also cover deep learning library limitations and possible optimisations (also see SI 5). Here, we have empirically evaluated the time and memory scaling of ESA against all 9 baselines. For a fair evaluation, we implement Flash attention for TokenGT as it was not originally supported. GraphGPS natively supports Flash attention, while Graphormer requires specific modifications to the attention matrix which are not currently supported.

We report the training time for a single epoch and the maximum allocated memory during training for QM9 and MNIST in Figure 4, and DOCKSTRING in Supplementary Figure 1. In terms of training time, GCN and GIN are consistently the fastest due to their simplicity and low number of parameters (also see Figure 6a). ESA is usually the next fastest, followed by other graph transformers. The strong GNN baselines, particularly PNA, and to an extent GATv2 and GAT, are among the slowest methods. In terms of memory, GCN and GIN are again the most efficient, followed by GraphGPS and TokenGT. ESA is only slightly behind, with PNA, DropGIN, GATv2, and GAT all being more memory intensive, particularly DropGIN.

We also order all methods according to their achieved performance and illustrate their rank relative to the total time spent training and the maximum memory allocated during training (Figure 5). ESA occupies the top left corner in the time plot, confirming its efficiency. The results are more spread out regarding memory, however the performance is still remarkable considering that ESA works over edges, whose number rapidly increases relative to the number of nodes that dictates the scaling of all other methods.

Finally, we report the number of parameters for all methods and their configurations (Figure 6a). Apart from GCN, GIN, and DropGIN, ESA has the lowest number of parameters. GAT and GATv2 have rapidly increasing parameter counts, likely due to the concatenation that happens between attention heads, while PNA also increases significantly quicker than ESA. Lastly, we demonstrate and discuss one of the bottlenecks
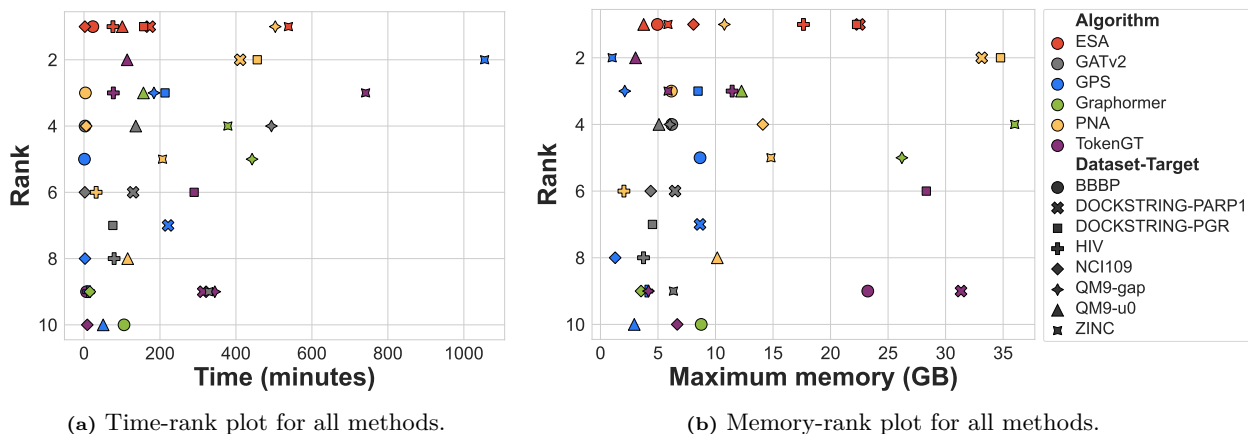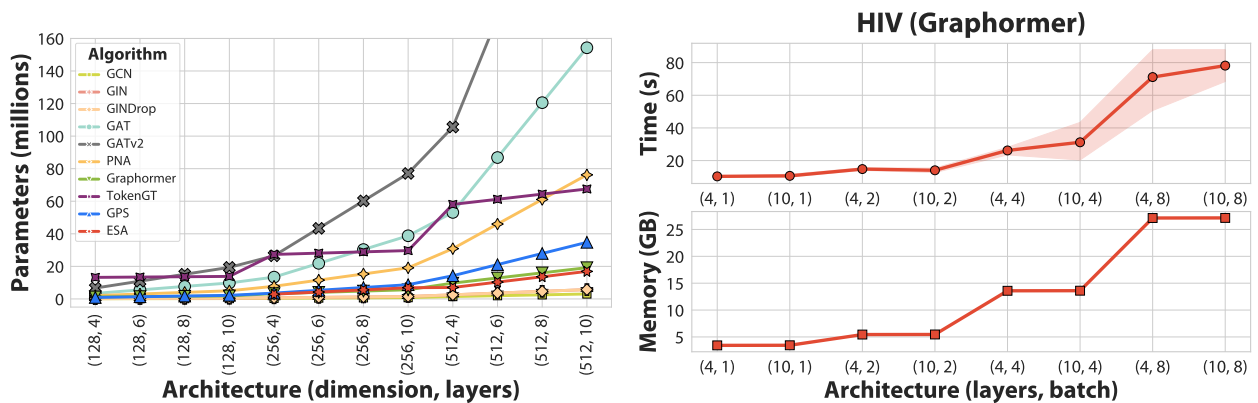
(a) Time-rank plot for all methods.

(b) Memory-rank plot for all methods.

Figure 5: All methods illustrated according to their achieved performance (rank) versus the total time spent training (minutes) in **Panel (a)** and the maximum allocated memory (GB) in **Panel (b)**.

(a) The number of parameters (in millions) for all the methods and different configurations in terms of the hidden dimension and the number of layers.

(b) The time and memory utilisation for a selection of Graphormer configurations. The varying parameters are the number of layers, and the number of input graphs.

Figure 6: The number of parameters for all methods in **Panel (a)** and an illustration of the time and memory bottleneck in Graphormer in **Panel (b)**. For Graphormer, even training on a single graph from HIV takes over 10 seconds and slightly under 5GB. Increasing the number of graphs to 8 and setting the batch size to 8 to ensure parallel computation increases the running time to around 80 seconds and slightly under 30GB. Extrapolating these numbers for a batch size of 8 to the entire dataset results in a training time of around 5 days for a single epoch.

in Graphormer for the dataset HIV ([Figure 6b](#)). This lack of efficiency is likely caused by the edge features computed by Graphormer, which depend quadratically on the number of nodes in the graph, and on the maximum shortest path in the graph.

5 Discussion

We presented an end-to-end attention-based approach for learning on graphs and demonstrated its effectiveness relative to tuned graph neural networks and recently proposed graph transformer architectures. The approach is free of expensive pre-processing steps and numerous other additional components designed to improve the inductive bias or expressive power (e.g., shortest paths, centrality, spatial and structural encodings, virtual nodes as readouts, expander graphs, transformations via interaction graphs, etc.). This shows huge potential and plenty of room for further fine-tuning and task-specific improvements. The interleaving operator inherent to ESA allows for vertical combination of masked and self-attention modules for learning effective token (i.e., edge or node) representations, leveraging relational information specified by input graphs while at the same time allowing to expand on this prior structure via self-attention. The recently released *Flex Attention* feature in PyTorch allows for further extensions via rich masking operators informed by graph structure while leveraging advancements in exact and efficient attention that have enabled our work.

Our comprehensive evaluation shows that the proposed approach consistently outperforms strong message passing baselines and recently proposed transformer-based approaches for learning on graphs. The takeaway from our extensive study is that the proposed approach is well suited for being a simple and yet extremely effective starting point for learning on graphs. Moreover, the approach has favourable computational complexity and scales better than strong GNNs and some graph transformers. The approach also does well in transfer learning settings, possibly paving the way for more research on foundational models for drug discovery, where the problem arises frequently in property prediction for expensive high-fidelity experiments.

References

1. Babai, L. & Kucera, L. *Canonical labelling of graphs in linear average time* in *Proceedings of the 20th Annual Symposium on Foundations of Computer Science* (1979), 39–46.

2. Morris, C. *et al.* Weisfeiler and Leman go machine learning: the story so far. *Journal of Machine Learning Research* **24** (2024).

3. Stokes, J. M. *et al.* A Deep Learning Approach to Antibiotic Discovery. *Cell* **180,** 688–702.e13. ISSN: 0092-8674 (2020).

4. Wong, F., Omori, S., Donghia, N. M., Zheng, E. J. & Collins, J. J. Discovering small-molecule senolytics with deep neural networks. *Nature Aging* **3,** 734–750. ISSN: 2662-8465 (June 2023).

5. Gasteiger, J., Groß, J. & Günnemann, S. *Directional Message Passing for Molecular Graphs* in *International Conference on Learning Representations (ICLR)* (2020).

6. Merchant, A. *et al.* Scaling deep learning for materials discovery. *Nature* **624,** 80–85. ISSN: 1476-4687 (Dec. 2023).

7. Baek, M. *et al.* Accurate prediction of protein structures and interactions using a three-track neural network. *Science* **373,** 871–876 (2021).

8. Dauparas, J. *et al.* Robust deep learning-based protein sequence design using ProteinMPNN. *Science* **378,** 49–56 (2022).

9. Chien, E. *et al.* Opportunities and challenges of graph neural networks in electrical engineering. *Nature Reviews Electrical Engineering* **1,** 529–546. ISSN: 2948-1201 (Aug. 2024).

10. Lam, R. *et al.* Learning skillful medium-range global weather forecasting. *Science* **382,** 1416–1421 (2023).

11. Corso, G., Cavalleri, L., Beaini, D., Liò, P. & Velickovic, P. *Principal Neighbourhood Aggregation for Graph Nets* in *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Curran Associates Inc., Vancouver, BC, Canada, 2020). ISBN: 9781713829546.

12. Buterez, D., Janet, J. P., Kiddle, S. J., Oglic, D. & Liò, P. *Graph Neural Networks with Adaptive Readouts* in *Advances in Neural Information Processing Systems* (eds Oh, A. H., Agarwal, A., Belgrave, D. & Cho, K.) (2022).

13. Buterez, D., Janet, J. P., Kiddle, S. J., Oglic, D. & Liò, P. Modelling local and general quantum mechanical properties with attention-based pooling. *Communications Chemistry* **6,** 262. ISSN: 2399-3669 (Nov. 2023).

14. Chen, K., Kunkel, C., Cheng, B., Reuter, K. & Margraf, J. T. Physics-inspired machine learning of localized intensive properties. *Chem. Sci.* **14,** 4913–4922 (18 2023).

15. Wagstaff, E., Fuchs, F., Engelcke, M., Posner, I. & Osborne, M. A. *On the Limitations of Representing Functions on Sets* in *Proceedings of the 36th International Conference on Machine Learning* **97** (2019), 6487–6494.

16. Li, Q., Han, Z. & Wu, X.-M. *Deeper insights into graph convolutional networks for semi-supervised learning* in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence* (2018).

17. Alon, U. & Yahav, E. *On the Bottleneck of Graph Neural Networks and its Practical Implications* in *International Conference on Learning Representations* (2021).

18. Godwin, J. *et al.* *Simple GNN Regularisation for 3D Molecular Property Prediction and Beyond* in *International Conference on Learning Representations* (2022).

19. Zhao, L. & Akoglu, L. *PairNorm: Tackling Oversmoothing in GNNs* in *International Conference on Learning Representations* (2020).

20. Cai, T. *et al.* *GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training* in *2021 International Conference on Machine Learning* (May 2021).

21. Hu, W. *et al.* *Strategies for Pre-training Graph Neural Networks* in *International Conference on Learning Representations* (2020).

22. Vaswani, A. *et al.* *Attention is All you Need* in *Advances in Neural Information Processing Systems* (eds Guyon, I. *et al.*) **30** (Curran Associates, Inc., 2017).

23. Veličković, P. *et al.* *Graph Attention Networks* in *International Conference on Learning Representations* (2018).

24. Brody, S., Alon, U. & Yahav, E. *How Attentive are Graph Attention Networks?* in *International Conference on Learning Representations* (2022).

25. Ying, C. *et al.* *Do Transformers Really Perform Badly for Graph Representation?* in *Thirty-Fifth Conference on Neural Information Processing Systems* (2021).

26. Kim, J. *et al.* *Pure Transformers are Powerful Graph Learners* in *Advances in Neural Information Processing Systems* (eds Oh, A. H., Agarwal, A., Belgrave, D. & Cho, K.) (2022).

27. Rampasek, L. *et al.* *Recipe for a General, Powerful, Scalable Graph Transformer* in *Advances in Neural Information Processing Systems* (eds Oh, A. H., Agarwal, A., Belgrave, D. & Cho, K.) (2022).

28. Shirzad, H., Velingker, A., Venkatachalam, B., Sutherland, D. J. & Sinop, A. K. *EXPHORMER: sparse transformers for graphs* in *Proceedings of the 40th International Conference on Machine Learning* (JMLR.org, 2023).

29. Tönshoff, J., Ritzert, M., Rosenbluth, E. & Grohe, M. *Where Did the Gap Go? Reassessing the Long-Range Graph Benchmark* in *The Second Learning on Graphs Conference* (2023).

30. Luo, Y., Shi, L. & Wu, X.-M. *Classic GNNs are Strong Baselines: Reassessing GNNs for Node Classification* 2024.

31. Ma, L. *et al. Graph inductive biases in transformers without message passing* in *Proceedings of the 40th International Conference on Machine Learning* (JMLR.org, Honolulu, Hawaii, USA, 2023).

32. Buterez, D., Janet, J. P., Kiddle, S. J., Oglic, D. & Lió, P. Transfer learning with graph neural networks for improved molecular property prediction in the multi-fidelity setting. *Nature Communications* **15,** 1517. ISSN: 2041-1723 (Feb. 2024).

33. Shi, Y. *et al. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification* in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21* (ed Zhou, Z.-H.) Main Track (International Joint Conferences on Artificial Intelligence Organization, Aug. 2021), 1548–1554.

34. Dwivedi, V. P. & Bresson, X. A Generalization of Transformer Networks to Graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications* (2021).

35. Kreuzer, D., Beaini, D., Hamilton, W. L., Létourneau, V. & Tossou, P. *Rethinking Graph Transformers with Spectral Attention* in *Advances in Neural Information Processing Systems* (eds Beygelzimer, A., Dauphin, Y., Liang, P. & Vaughan, J. W.) (2021).

36. Min, E. *et al. Neighbour Interaction based Click-Through Rate Prediction via Graph-masked Transformer* in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Association for Computing Machinery, Madrid, Spain, 2022), 353–362. ISBN: 9781450387323.

37. Rampášek, L. *et al.* Recipe for a General, Powerful, Scalable Graph Transformer. *Advances in Neural Information Processing Systems* **35** (2022).

38. Choromanski, K. M. *et al. Rethinking Attention with Performers* in *International Conference on Learning Representations* (2021).

39. Lee, J. *et al. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks* in *Proceedings of the 36th International Conference on Machine Learning* (2019), 3744–3753.

40. Xiong, R. *et al. On Layer Normalization in the Transformer Architecture* in *Proceedings of the 37th International Conference on Machine Learning* (JMLR.org, 2020).

41. Shazeer, N. GLU Variants Improve Transformer (2020).

42. Rabe, M. N. & Staats, C. Self-attention Does Not Need $O(n^2)$ Memory (2021).

43. Dao, T., Fu, D. Y., Ermon, S., Rudra, A. & Ré, C. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness* in *Advances in Neural Information Processing Systems* (2022).

44. Dao, T. *FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning* 2023.

45. Shah, J. *et al. FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision* 2024.

46. Paszke, A. *et al.* in *Advances in Neural Information Processing Systems 32* 8024–8035 (Curran Associates, Inc., 2019).

47. Lefaudeux, B. *et al. xFormers: A modular and hackable Transformer modelling library* https://github.com/facebookresearch/xformers. 2022.

48. Ramakrishnan, R., Dral, P. O., Rupp, M. & von Lilienfeld, O. A. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data* **1,** 140022. ISSN: 2052-4463 (Aug. 2014).

49. García-Ortegón, M. *et al.* DOCKSTRING: Easy Molecular Docking Yields Better Benchmarks for Ligand Design. *Journal of Chemical Information and Modeling* **62.** PMID: 35849793, 3486–3502 (2022).

50. Xiong, Z. *et al.* Pushing the Boundaries of Molecular Representation for Drug Discovery with the Graph Attention Mechanism. *Journal of Medicinal Chemistry* **63.** PMID: 31408336, 8749–8760 (2020).

51. Hu, W. *et al.* OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs. *arXiv preprint arXiv:2103.09430* (2021).

52. Choi, Y. Y., Park, S. W., Lee, M. & Woo, Y. *Topology-Informed Graph Transformer* 2024.

53. Hussain, M. S., Zaki, M. J. & Subramanian, D. *Triplet Interaction Improves Graph Transformers: Accurate Molecular Graph Learning with Triplet Graph Transformers* in *Forty-first International Conference on Machine Learning* (2024).

54. Lu, S., Gao, Z., He, D., Zhang, L. & Ke, G. Data-driven quantum chemical property prediction leveraging 3D conformations with Uni-Mol+. *Nature Communications* **15,** 7104. ISSN: 2041-1723 (Aug. 2024).

55. Luo, S. *et al. One Transformer Can Understand Both 2D & 3D Molecular Data* in *The Eleventh International Conference on Learning Representations* (2023).

56. Jin, W., Barzilay, R. & Jaakkola, T. *Junction Tree Variational Autoencoder for Molecular Graph Generation* in *Proceedings of the 35th International Conference on Machine Learning* (eds Dy, J. & Krause, A.) **80** (PMLR, July 2018), 2323–2332.

57. Kondratyev, V., Dryzhakov, M., Gimadiev, T. & Slutskiy, D. Generative model based on junction tree variational autoencoder for HOMO value prediction and molecular optimization. *Journal of Cheminformatics* **15,** 11. ISSN: 1758-2946 (Feb. 2023).

58. Dwivedi, V. P. *et al.* Benchmarking graph neural networks. *J. Mach. Learn. Res.* **24.** ISSN: 1532-4435 (Mar. 2024).

59. Dwivedi, V. P. *et al. Long Range Graph Benchmark* in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track* (2022).

60. Zitnick, C. L. *et al. An Introduction to Electrocatalyst Design using Machine Learning for Renewable Energy Storage* 2020.

61. Chanussot, L. *et al.* Open Catalyst 2020 (OC20) Dataset and Community Challenges. *ACS Catalysis* **11,** 6059–6072 (2021).

62. Shi, Y. *et al. Benchmarking Graphormer on Large-Scale Molecular Modeling Datasets* 2023.

63. Fediai, A., Reiser, P., Peña, J. E. O., Friederich, P. & Wenzel, W. Accurate GW frontier orbital energies of 134 kilo molecules. *Scientific Data* **10,** 581. ISSN: 2052-4463 (Sept. 2023).

64. Chen, D., Schulz, T. H. & Borgwardt, K. *Learning Long Range Dependencies on Graphs via Random Walks* 2024.

65. Cai, C. & Wang, Y. A simple yet effective baseline for non-attribute graph classification. *CoRR* **abs/1811.03508** (2018).

66. Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A. & Prokhorenkova, L. *A critical look at the evaluation of GNNs under heterophily: Are we really making progress?* in *The Eleventh International Conference on Learning Representations* (2023).

67. Hamilton, W. L., Ying, R. & Leskovec, J. *Inductive Representation Learning on Large Graphs* in *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Curran Associates Inc., Long Beach, California, USA, 2017), 1025–1035. ISBN: 9781510860964.

68. Loshchilov, I. & Hutter, F. *Decoupled Weight Decay Regularization* in *International Conference on Learning Representations* (2019).

69. Dettmers, T., Lewis, M., Shleifer, S. & Zettlemoyer, L. *8-bit Optimizers via Block-wise Quantization* in *International Conference on Learning Representations* (2022).

70. Wolf, T. *et al. Transformers: State-of-the-Art Natural Language Processing* in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (eds Liu, Q. & Schlangen, D.) (Association for Computational Linguistics, Online, Oct. 2020), 38–45.

71. Chicco, D. & Jurman, G. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics* **21,** 6. ISSN: 1471-2164 (Jan. 2020).

72. Chicco, D., Warrens, M. J. & Jurman, G. The Matthews Correlation Coefficient (MCC) is More Informative Than Cohen's Kappa and Brier Score in Binary Classification Assessment. *IEEE Access* **9,** 78368–78381 (2021).

73. Chicco, D., Tötsch, N. & Jurman, G. The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. *BioData Mining* **14,** 13. ISSN: 1756-0381 (Feb. 2021).

74. Stoica, P. & Babu, P. Pearson–Matthews correlation coefficients for binary and multinary classification. *Signal Processing* **222,** 109511. ISSN: 0165-1684 (2024).

75. Chicco, D. & Jurman, G. The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification. *BioData Mining* **16,** 4. ISSN: 1756-0381 (Feb. 2023).

76. Hand, D. J. *Mismatched models, wrong results, and dreadful decisions: on choosing appropriate data mining tools* in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Association for Computing Machinery, Paris, France, 2009), 1–2. ISBN: 9781605584959.

77. Kwegyir-Aggrey, K., Gerchick, M., Mohan, M., Horowitz, A. & Venkatasubramanian, S. *The Misuse of AUC: What High Impact Risk Assessment Gets Wrong* in *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency* (Association for Computing Machinery, Chicago, IL, USA, 2023), 1570–1583. ISBN: 9798400701924.

78. Chicco, D., Warrens, M. J. & Jurman, G. The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. en. *PeerJ Comput Sci* **7,** e623 (July 2021).

79. Wang, Q., Chen, D. Z., Wijesinghe, A., Li, S. & Farhan, M. $\mathcal{N}$*-WL: A New Hierarchy of Expressivity for Graph Neural Networks* in *The Eleventh International Conference on Learning Representations* (2023).

SI 1 Additional benchmarking results

Supplementary Table 1: The root mean squared error on QM9 (RMSE is the standard for quantum mechanics) and the R^2 for DOCKSTRING (DOCK) and MOLECULENET (MN), presented as mean ± standard deviation over 5 runs, and including GCN and GIN. The mean absolute error (MAE) is reported for PCQM4MV2 over a single run due to the size of the dataset and the field standards. The lowest MAEs and RMSEs and highest R^2 values are highlighted in bold. OOM denotes out-of-memory errors.

	Target	GCN	GIN	DropGIN	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	ESA
QM9 (↓)	μ	0.629 ± 0.005	0.550 ± 0.011	0.552 ± 0.010	0.552 ± 0.011	0.545 ± 0.008	**0.535 ± 0.008**	0.627 ± 0.014	0.755 ± 0.025	0.945 ± 0.174	0.564 ± 0.004
	α	0.525 ± 0.051	0.438 ± 0.020	0.445 ± 0.025	0.479 ± 0.020	0.460 ± 0.018	0.476 ± 0.070	0.404 ± 0.017	0.447 ± 0.014	0.605 ± 0.130	**0.398 ± 0.003**
	ϵ_{HOMO}	0.128 ± 0.005	0.104 ± 0.001	0.106 ± 0.002	0.107 ± 0.002	0.104 ± 0.001	**0.099 ± 0.001**	0.107 ± 0.002	0.123 ± 0.003	0.113 ± 0.003	0.103 ± 0.003
	ϵ_{LUMO}	0.136 ± 0.002	0.120 ± 0.004	0.123 ± 0.004	0.124 ± 0.003	0.127 ± 0.001	**0.109 ± 0.001**	0.112 ± 0.001	0.130 ± 0.003	0.108 ± 0.002	0.114 ± 0.001
	$\Delta\epsilon$	0.184 ± 0.006	0.164 ± 0.004	0.166 ± 0.004	0.163 ± 0.006	0.163 ± 0.003	**0.139 ± 0.001**	0.163 ± 0.004	0.177 ± 0.006	0.154 ± 0.006	0.152 ± 0.001
	$\langle R^2 \rangle$	33.913 ± 0.922	29.925 ± 0.302	29.870 ± 0.343	30.911 ± 0.152	30.149 ± 0.360	28.503 ± 0.444	29.628 ± 0.351	31.540 ± 0.422	30.421 ± 1.192	**28.328 ± 0.321**
	ZPVE	0.035 ± 0.001	0.032 ± 0.001	0.034 ± 0.002	0.033 ± 0.001	0.032 ± 0.001	0.032 ± 0.003	0.059 ± 0.031	0.030 ± 0.001	0.033 ± 0.006	**0.026 ± 0.001**
	U_0	30.971 ± 6.472	42.024 ± 17.272	29.817 ± 6.322	27.823 ± 3.981	25.405 ± 3.617	31.644 ± 6.493	24.600 ± 4.699	15.477 ± 4.675	14.496 ± 4.342	**4.777 ± 0.671**
	U	25.276 ± 4.471	25.098 ± 5.972	24.741 ± 6.311	30.914 ± 2.620	24.919 ± 4.183	25.038 ± 5.479	15.546 ± 8.228	12.449 ± 1.864	15.820 ± 4.262	**6.799 ± 2.809**
	H	30.924 ± 5.369	24.746 ± 2.314	34.019 ± 5.663	28.924 ± 4.108	23.422 ± 2.767	27.338 ± 8.167	19.006 ± 4.645	11.418 ± 3.716	12.923 ± 4.287	**6.018 ± 1.324**
	G	26.138 ± 10.078	26.942 ± 2.910	25.412 ± 4.863	31.494 ± 1.857	23.240 ± 2.535	22.308 ± 3.453	31.198 ± 4.614	29.721 ± 8.790	13.081 ± 4.806	**6.104 ± 1.334**
	c_{V}	0.221 ± 0.014	0.187 ± 0.008	0.195 ± 0.012	0.191 ± 0.005	0.190 ± 0.002	0.179 ± 0.009	0.177 ± 0.023	0.180 ± 0.004	0.190 ± 0.027	**0.158 ± 0.001**
	U_0^{ATOM}	0.371 ± 0.017	0.391 ± 0.035	0.378 ± 0.022	0.392 ± 0.035	0.384 ± 0.021	0.353 ± 0.018	0.289 ± 0.004	0.304 ± 0.021	0.330 ± 0.051	**0.241 ± 0.006**
	U^{ATOM}	0.390 ± 0.029	0.377 ± 0.029	0.404 ± 0.044	0.483 ± 0.072	0.397 ± 0.036	0.361 ± 0.020	0.302 ± 0.014	0.302 ± 0.004	0.334 ± 0.058	**0.243 ± 0.005**
	H^{ATOM}	0.396 ± 0.025	0.370 ± 0.026	0.376 ± 0.038	0.383 ± 0.021	0.406 ± 0.062	0.373 ± 0.029	0.301 ± 0.012	0.312 ± 0.014	0.363 ± 0.089	**0.245 ± 0.004**
	G^{ATOM}	0.368 ± 0.008	0.385 ± 0.021	0.372 ± 0.036	0.342 ± 0.018	0.329 ± 0.009	0.314 ± 0.021	0.261 ± 0.005	0.273 ± 0.006	0.360 ± 0.075	**0.225 ± 0.015**
	A	0.979 ± 0.049	1.317 ± 0.386	0.904 ± 0.059	0.972 ± 0.122	1.078 ± 0.161	1.007 ± 0.101	64.877 ± 29.771	3.823 ± 2.052	1.422 ± 0.437	**0.746 ± 0.106**
	B	0.295 ± 0.004	0.196 ± 0.047	0.194 ± 0.050	0.211 ± 0.038	0.264 ± 0.010	0.256 ± 0.024	0.102 ± 0.028	0.109 ± 0.013	0.158 ± 0.052	**0.079 ± 0.011**
	C	0.284 ± 0.001	0.167 ± 0.062	0.269 ± 0.016	0.266 ± 0.006	0.276 ± 0.004	0.277 ± 0.012	0.115 ± 0.037	0.097 ± 0.025	0.124 ± 0.046	**0.050 ± 0.012**
DOCK (↑)	ESR2	0.642 ± 0.003	0.668 ± 0.003	0.675 ± 0.003	0.666 ± 0.002	0.655 ± 0.004	0.696 ± 0.002	OOM	0.641 ± 0.008	0.676 ± 0.002	**0.697 ± 0.001**
	F2	0.878 ± 0.001	0.887 ± 0.002	0.886 ± 0.001	0.886 ± 0.001	0.885 ± 0.002	0.891 ± 0.002	OOM	0.872 ± 0.006	0.879 ± 0.004	**0.891 ± 0.000**
	KIT	0.814 ± 0.002	0.833 ± 0.001	0.835 ± 0.002	0.833 ± 0.000	0.826 ± 0.001	**0.843 ± 0.001**	OOM	0.800 ± 0.009	0.832 ± 0.001	0.841 ± 0.001
	PARP1	0.912 ± 0.001	0.922 ± 0.001	0.920 ± 0.002	0.921 ± 0.001	0.919 ± 0.001	0.924 ± 0.001	OOM	0.907 ± 0.005	0.915 ± 0.005	**0.925 ± 0.000**
	PGR	0.658 ± 0.004	0.696 ± 0.001	0.702 ± 0.002	0.681 ± 0.005	0.666 ± 0.006	0.717 ± 0.003	OOM	0.684 ± 0.010	0.703 ± 0.009	**0.725 ± 0.003**
MN (↑)	FSOLV	0.957 ± 0.008	0.964 ± 0.007	0.972 ± 0.005	0.959 ± 0.009	0.970 ± 0.007	0.951 ± 0.008	0.927 ± 0.005	0.930 ± 0.016	0.861 ± 0.032	**0.977 ± 0.001**
	LIPO	0.800 ± 0.007	0.819 ± 0.006	0.809 ± 0.007	0.820 ± 0.012	0.821 ± 0.008	**0.830 ± 0.006**	0.607 ± 0.043	0.545 ± 0.022	0.790 ± 0.004	0.809 ± 0.007
	ESOL	0.936 ± 0.005	0.938 ± 0.010	0.935 ± 0.011	0.930 ± 0.006	0.928 ± 0.005	0.942 ± 0.006	0.908 ± 0.018	0.892 ± 0.032	0.911 ± 0.003	**0.944 ± 0.002**

Supplementary Table 2: The mean absolute error (MAE) on ZINC, presented as mean ± standard deviation over 5 runs. The lowest values are highlighted in bold.

Dataset (↓)	GCN	GIN	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	ESA	ESA (PE)
ZINC	0.152 ± 0.02	0.068 ± 0.00	0.078 ± 0.01	0.079 ± 0.00	0.057 ± 0.01	0.036 ± 0.00	0.047 ± 0.01	0.024 ± 0.01	0.027 ± 0.00	$\mathbf{0.017 \pm 0.00}$

Supplementary Table 3: Transfer learning performance (RMSE) on QM9 for HOMO and LUMO, presented as mean ± standard deviation over 5 different runs, including GCN and GIN. All models use the 3D atomic coordinates and atom types as inputs and no other node or edge features. 'Strat.' stands for strategy and specifies the type of learning: GW only (no transfer learning), inductive, or transductive. The lowest values are highlighted in bold.

Task	Strat.	GCN	GIN	DropGIN	GAT	GATv2	PNA	Grph.	TokenGT	GPS	ESA
HOMO	GW	0.171 ± 0.004	0.162 ± 0.002	0.162 ± 0.002	0.159 ± 0.003	0.157 ± 0.002	$\mathbf{0.151 \pm 0.001}$	0.179 ± 0.009	0.200 ± 0.008	0.162 ± 0.002	0.152 ± 0.003
	Ind.	0.143 ± 0.004	0.138 ± 0.001	0.136 ± 0.000	0.131 ± 0.001	0.133 ± 0.003	0.132 ± 0.002	0.134 ± 0.000	0.156 ± 0.000	0.151 ± 0.002	$\mathbf{0.131 \pm 0.000}$
	Trans.	0.131 ± 0.001	0.125 ± 0.001	0.126 ± 0.002	0.123 ± 0.001	0.124 ± 0.001	0.121 ± 0.001	0.125 ± 0.001	0.137 ± 0.000	0.147 ± 0.002	$\mathbf{0.119 \pm 0.000}$
LUMO	GW	0.181 ± 0.002	0.180 ± 0.002	0.180 ± 0.002	0.181 ± 0.002	0.178 ± 0.002	0.174 ± 0.004	0.190 ± 0.006	0.204 ± 0.006	0.178 ± 0.002	$\mathbf{0.174 \pm 0.001}$
	Ind.	0.161 ± 0.001	0.159 ± 0.001	0.159 ± 0.001	0.156 ± 0.001	0.157 ± 0.001	0.156 ± 0.002	0.151 ± 0.001	0.165 ± 0.000	0.167 ± 0.001	$\mathbf{0.150 \pm 0.001}$
	Trans.	0.159 ± 0.002	0.155 ± 0.001	0.157 ± 0.001	0.153 ± 0.001	0.153 ± 0.001	0.153 ± 0.001	0.147 ± 0.001	0.156 ± 0.000	0.169 ± 0.001	$\mathbf{0.146 \pm 0.000}$

Supplementary Table 4: Matthews correlation coefficient (MCC) for graph-level molecular classification tasks – MoleculeNet and National Cancer Institute (NCI), presented as mean ± standard deviation over 5 different runs, and including GCN and GIN. OOM denotes out-of-memory errors. The highest mean values are highlighted in bold.

	Data (↑)	GCN	GIN	DropGIN	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	ESA
MN	BBBP	0.674 ± 0.034	0.704 ± 0.028	0.685 ± 0.017	0.744 ± 0.012	0.728 ± 0.032	0.731 ± 0.028	0.552 ± 0.012	0.578 ± 0.065	0.705 ± 0.044	$\mathbf{0.835 \pm 0.014}$
	BACE	0.631 ± 0.028	0.646 ± 0.013	0.654 ± 0.034	0.632 ± 0.018	0.645 ± 0.026	0.638 ± 0.017	0.522 ± 0.020	0.578 ± 0.033	0.618 ± 0.032	$\mathbf{0.721 \pm 0.019}$
	HIV	0.448 ± 0.035	0.408 ± 0.060	0.458 ± 0.028	0.421 ± 0.061	0.337 ± 0.059	0.417 ± 0.045	OOM	0.455 ± 0.017	0.247 ± 0.211	$\mathbf{0.533 \pm 0.012}$
NCI	NCI1	0.682 ± 0.013	0.694 ± 0.017	0.686 ± 0.027	0.701 ± 0.018	0.646 ± 0.029	0.697 ± 0.025	0.540 ± 0.025	0.532 ± 0.034	0.697 ± 0.027	$\mathbf{0.755 \pm 0.012}$
	NCI109	0.665 ± 0.024	0.684 ± 0.015	0.681 ± 0.021	0.658 ± 0.008	0.664 ± 0.015	0.670 ± 0.018	0.504 ± 0.022	0.453 ± 0.029	0.623 ± 0.014	$\mathbf{0.700 \pm 0.010}$

Supplementary Table 5: Accuracy for graph-level molecular classification tasks – MoleculeNet and National Cancer Institute (NCI), presented as mean ± standard deviation over 5 different runs, and including GCN and GIN. OOM denotes out-of-memory errors. The highest mean values are highlighted in bold.

	Data (↑)	GCN	GIN	DropGIN	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	ESA
MN	BBBP	0.871 ± 0.012	0.882 ± 0.010	0.875 ± 0.006	0.898 ± 0.004	0.892 ± 0.012	0.893 ± 0.010	0.826 ± 0.005	0.832 ± 0.022	0.883 ± 0.017	$\mathbf{0.932 \pm 0.007}$
	BACE	0.813 ± 0.014	0.820 ± 0.007	0.824 ± 0.017	0.813 ± 0.009	0.820 ± 0.012	0.817 ± 0.009	0.758 ± 0.011	0.786 ± 0.015	0.804 ± 0.017	$\mathbf{0.858 \pm 0.010}$
	HIV	0.974 ± 0.001	0.973 ± 0.001	0.974 ± 0.001	0.973 ± 0.002	0.971 ± 0.001	0.973 ± 0.001	OOM	0.973 ± 0.001	0.971 ± 0.003	$\mathbf{0.976 \pm 0.001}$
NCI	NCI1	0.842 ± 0.006	0.848 ± 0.008	0.843 ± 0.014	0.851 ± 0.010	0.824 ± 0.015	0.850 ± 0.012	0.770 ± 0.012	0.767 ± 0.018	0.850 ± 0.014	$\mathbf{0.878 \pm 0.006}$
	NCI109	0.831 ± 0.011	0.842 ± 0.007	0.840 ± 0.010	0.826 ± 0.005	0.831 ± 0.007	0.834 ± 0.009	0.749 ± 0.011	0.721 ± 0.017	0.809 ± 0.006	$\mathbf{0.850 \pm 0.005}$

Supplementary Table 6: Matthews correlation coefficient (MCC) for graph-level classification tasks from various domains, presented as mean ± standard deviation over 5 different runs, and including GCN and GIN. OOM denotes out-of-memory errors, and N/A that the model is unavailable (e.g., node/edge features are not integers, which are required for Graphormer and TokenGT). The highest mean values are highlighted in bold.

	Dataset (↑)	GCN	GIN	DropGIN	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	ESA
	MALNETTINY	0.896 ± 0.006	0.903 ± 0.005	0.902 ± 0.006	0.899 ± 0.006	0.902 ± 0.007	0.915 ± 0.008	OOM	0.777 ± 0.008	0.795 ± 0.011	**0.931 ± 0.001**
VIS.	MNIST	0.957 ± 0.002	0.965 ± 0.004	0.970 ± 0.001	0.972 ± 0.002	0.979 ± 0.002	0.978 ± 0.003	N/A	N/A	0.980 ± 0.001	**0.986 ± 0.000**
VIS.	CIFAR10	0.621 ± 0.003	0.614 ± 0.006	0.614 ± 0.009	0.659 ± 0.009	0.662 ± 0.011	0.686 ± 0.005	N/A	N/A	0.708 ± 0.005	**0.727 ± 0.003**
BIO.	ENZYMES	0.695 ± 0.048	0.632 ± 0.046	0.576 ± 0.039	0.748 ± 0.020	0.744 ± 0.023	0.684 ± 0.034	N/A	N/A	0.734 ± 0.045	**0.751 ± 0.009**
BIO.	PROTEINS	0.419 ± 0.037	0.421 ± 0.036	0.459 ± 0.005	0.463 ± 0.036	0.490 ± 0.042	0.467 ± 0.067	N/A	N/A	0.443 ± 0.022	**0.589 ± 0.017**
BIO.	DD	0.546 ± 0.058	0.539 ± 0.032	0.537 ± 0.070	0.465 ± 0.049	0.530 ± 0.034	0.559 ± 0.080	OOM	0.459 ± 0.049	0.605 ± 0.041	**0.652 ± 0.030**
SYNTH	SYNTH	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	N/A	N/A	1.000 ± 0.000	**1.000 ± 0.000**
SYNTH	SYNT N.	0.699 ± 0.029	0.910 ± 0.030	0.975 ± 0.051	0.761 ± 0.067	0.909 ± 0.067	1.000 ± 0.000	N/A	N/A	1.000 ± 0.000	**1.000 ± 0.000**
SYNTH	SYNTHIE	0.935 ± 0.054	0.930 ± 0.045	0.942 ± 0.024	0.701 ± 0.045	0.798 ± 0.038	0.879 ± 0.058	N/A	N/A	**0.951 ± 0.019**	0.947 ± 0.016
SOCIAL	IMDB-B	0.603 ± 0.056	0.537 ± 0.199	0.608 ± 0.061	0.688 ± 0.037	0.600 ± 0.049	0.563 ± 0.067	0.565 ± 0.045	0.606 ± 0.052	0.598 ± 0.045	**0.738 ± 0.026**
SOCIAL	IMDB-M	0.216 ± 0.044	0.119 ± 0.079	0.117 ± 0.099	0.203 ± 0.052	0.202 ± 0.035	0.034 ± 0.068	0.222 ± 0.024	0.202 ± 0.023	0.232 ± 0.021	**0.247 ± 0.034**
SOCIAL	TWITCH E.	0.386 ± 0.006	0.358 ± 0.023	0.379 ± 0.011	0.373 ± 0.011	0.371 ± 0.011	0.078 ± 0.159	0.387 ± 0.003	0.393 ± 0.001	0.395 ± 0.001	**0.398 ± 0.000**
SOCIAL	REDDIT THR.	0.556 ± 0.007	0.556 ± 0.011	0.556 ± 0.007	0.533 ± 0.021	0.536 ± 0.023	0.113 ± 0.227	0.567 ± 0.003	0.564 ± 0.001	**0.568 ± 0.003**	0.568 ± 0.002

Supplementary Table 7: Accuracy for graph-level classification tasks from various domains, presented as mean ± standard deviation over 5 different runs, and including GCN and GIN. OOM denotes out-of-memory errors, and N/A that the model is unavailable (e.g., node/edge features are not integers, which are required for Graphormer and TokenGT). The highest mean values are highlighted in bold.

	Dataset (↑)	GCN	GIN	DropGIN	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	ESA
	MALNETTINY	0.916 ± 0.005	0.922 ± 0.004	0.921 ± 0.005	0.918 ± 0.005	0.921 ± 0.006	0.931 ± 0.006	OOM	0.820 ± 0.007	0.835 ± 0.009	**0.944 ± 0.001**
Vis.	MNIST	0.961 ± 0.001	0.969 ± 0.004	0.973 ± 0.001	0.975 ± 0.002	0.981 ± 0.001	0.980 ± 0.003	N/A	N/A	0.982 ± 0.001	**0.988 ± 0.000**
	CIFAR10	0.659 ± 0.003	0.652 ± 0.005	0.652 ± 0.008	0.693 ± 0.008	0.695 ± 0.010	0.717 ± 0.005	N/A	N/A	0.737 ± 0.005	**0.754 ± 0.002**
Bio.	ENZYMES	0.735 ± 0.039	0.683 ± 0.037	0.651 ± 0.037	0.786 ± 0.014	0.780 ± 0.019	0.730 ± 0.022	N/A	N/A	0.777 ± 0.039	**0.794 ± 0.015**
	PROTEINS	0.755 ± 0.015	0.755 ± 0.017	0.768 ± 0.000	0.768 ± 0.015	0.777 ± 0.020	0.777 ± 0.029	N/A	N/A	0.768 ± 0.011	**0.827 ± 0.007**
	DD	0.782 ± 0.031	0.773 ± 0.020	0.782 ± 0.033	0.731 ± 0.031	0.760 ± 0.020	0.790 ± 0.039	OOM	0.739 ± 0.030	0.808 ± 0.017	**0.835 ± 0.016**
Synth	SYNTH	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	N/A	N/A	1.000 ± 0.000	**1.000 ± 0.000**
	SYNT N.	0.847 ± 0.016	0.953 ± 0.016	0.987 ± 0.027	0.880 ± 0.034	0.953 ± 0.034	1.000 ± 0.000	N/A	N/A	1.000 ± 0.000	**1.000 ± 0.000**
	SYNTHIE	0.956 ± 0.039	0.955 ± 0.033	0.963 ± 0.018	0.776 ± 0.037	0.855 ± 0.031	0.918 ± 0.040	N/A	N/A	0.958 ± 0.014	**0.963 ± 0.012**
Social	IMDB-B	0.802 ± 0.028	0.766 ± 0.097	0.804 ± 0.030	0.842 ± 0.018	0.800 ± 0.024	0.780 ± 0.034	0.780 ± 0.023	0.802 ± 0.026	0.794 ± 0.024	**0.868 ± 0.013**
	IMDB-M	0.476 ± 0.030	0.411 ± 0.052	0.410 ± 0.064	0.470 ± 0.033	0.469 ± 0.024	0.356 ± 0.046	0.484 ± 0.015	0.470 ± 0.015	0.476 ± 0.013	**0.487 ± 0.011**
	TWITCH E.	0.697 ± 0.003	0.682 ± 0.012	0.694 ± 0.005	0.690 ± 0.005	0.690 ± 0.005	0.506 ± 0.098	0.698 ± 0.001	0.701 ± 0.001	0.702 ± 0.000	**0.703 ± 0.000**
	REDDIT THR.	0.778 ± 0.003	0.776 ± 0.005	0.777 ± 0.003	0.765 ± 0.010	0.767 ± 0.011	0.545 ± 0.118	0.782 ± 0.001	0.780 ± 0.001	**0.783 ± 0.001**	0.782 ± 0.001

Supplementary Table 8: Matthews correlation coefficient (MCC) for 11 node-level classification tasks, presented as mean ± standard deviation over 5 different runs, including GIN and DropGIN. The number of nodes for the shortest path (SP) benchmarks is given in parentheses (based on randomly-generated 'infected' Erdős–Rényi (ER) graphs; SI 8 for details). Additional heterophily results are provided in Supplementary Table 10. The highest mean values are highlighted in bold.

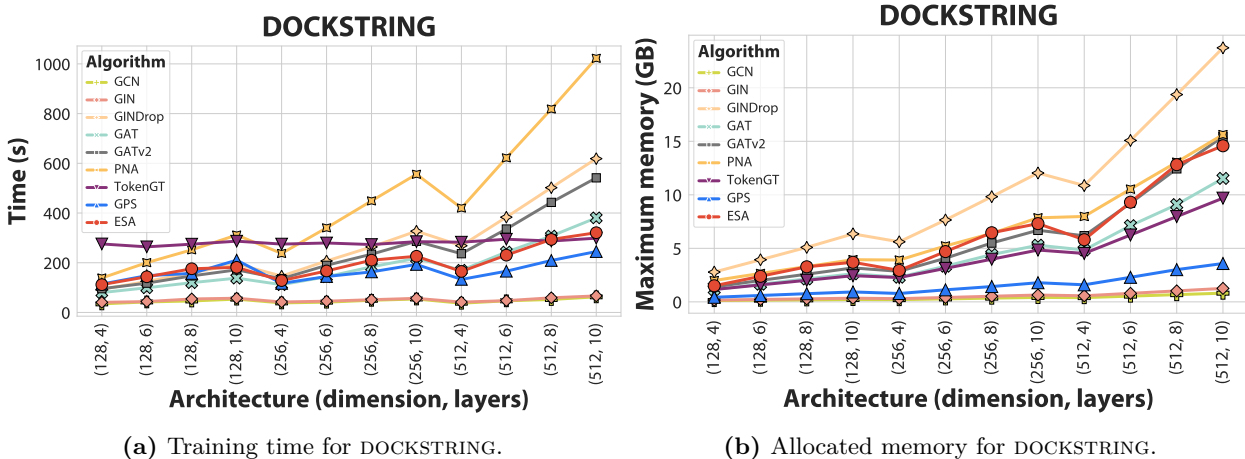|	Dataset (↑)	GCN	GIN	DropGIN	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	ESA
	PPI	0.979 ± 0.002	0.905 ± 0.066	0.785 ± 0.141	**0.990 ± 0.000**	0.982 ± 0.005	0.990 ± 0.000	N/A	N/A	N/A	0.989 ± 0.001
CITE	CITESEER	0.608 ± 0.008	0.587 ± 0.010	0.324 ± 0.017	0.587 ± 0.030	0.613 ± 0.006	0.511 ± 0.033	OOM	0.384 ± 0.022	0.538 ± 0.012	**0.632 ± 0.005**
	CORA	0.767 ± 0.008	0.727 ± 0.010	0.490 ± 0.033	0.748 ± 0.014	0.727 ± 0.010	0.637 ± 0.029	OOM	0.366 ± 0.185	0.643 ± 0.039	**0.768 ± 0.005**
HETEROPHILY	ROMAN EMP.	0.470 ± 0.004	0.791 ± 0.003	0.796 ± 0.002	0.741 ± 0.010	0.763 ± 0.004	0.855 ± 0.002	N/A	N/A	0.837 ± 0.014	**0.869 ± 0.002**
	AMAZON R.	0.179 ± 0.003	0.262 ± 0.015	0.228 ± 0.018	0.256 ± 0.009	0.253 ± 0.013	0.206 ± 0.018	N/A	N/A	0.114 ± 0.139	**0.336 ± 0.006**
	MINESWEEPER	0.303 ± 0.002	0.563 ± 0.028	0.455 ± 0.162	0.477 ± 0.018	0.512 ± 0.010	0.624 ± 0.043	N/A	N/A	0.564 ± 0.011	**0.688 ± 0.001**
	TOLOKERS	0.299 ± 0.011	0.299 ± 0.065	0.340 ± 0.012	0.384 ± 0.009	0.386 ± 0.005	0.350 ± 0.048	N/A	N/A	0.351 ± 0.021	**0.427 ± 0.004**
	SQUIRREL	0.197 ± 0.011	0.194 ± 0.026	0.230 ± 0.019	0.237 ± 0.017	0.238 ± 0.013	0.224 ± 0.011	0.104 ± 0.085	0.175 ± 0.017	0.228 ± 0.021	**0.289 ± 0.006**
	CHAMELEON	0.324 ± 0.006	0.272 ± 0.024	0.287 ± 0.026	0.240 ± 0.056	0.281 ± 0.030	0.267 ± 0.030	0.253 ± 0.038	0.260 ± 0.045	0.298 ± 0.081	**0.387 ± 0.018**
INF	ER (15K)	0.216 ± 0.016	0.387 ± 0.067	0.244 ± 0.049	0.315 ± 0.001	0.316 ± 0.000	0.543 ± 0.086	OOM	0.065 ± 0.000	0.178 ± 0.040	**0.915 ± 0.007**
	ER (30K)	0.094 ± 0.035	0.330 ± 0.021	0.292 ± 0.022	0.102 ± 0.064	0.102 ± 0.058	0.423 ± 0.049	OOM	OOM	OOM	**0.872 ± 0.008**

Supplementary Table 9: Accuracy for 11 node-level classification tasks, presented as mean ± standard deviation over 5 different runs, including GIN and DropGIN. The number of nodes for the shortest path (SP) benchmarks is given in parentheses (based on randomly-generated 'infected' Erdős–Rényi (ER) graphs; SI 8 for details). Additional heterophily results are provided in Supplementary Table 10. The highest mean values are highlighted in bold.

	Dataset (↑)	GCN	GIN	DropGIN	GAT	GATv2	PNA	Graphormer	TokenGT	GPS	ESA
	PPI	0.991 ± 0.001	0.960 ± 0.027	0.913 ± 0.056	**0.996 ± 0.000**	0.992 ± 0.002	0.996 ± 6.480	N/A	N/A	N/A	0.995 ± 0.000
CITE	CITESEER	0.648 ± 0.008	0.626 ± 0.008	0.426 ± 0.016	0.625 ± 0.018	0.649 ± 0.005	0.557 ± 0.027	OOM	0.470 ± 0.016	0.614 ± 0.011	**0.651 ± 0.008**
	CORA	**0.822 ± 0.005**	0.786 ± 0.012	0.595 ± 0.029	0.803 ± 0.014	0.785 ± 0.007	0.716 ± 0.019	OOM	0.456 ± 0.159	0.698 ± 0.033	0.820 ± 0.004
HETEROPHILY	ROMAN EMP.	0.462 ± 0.002	0.755 ± 0.005	0.764 ± 0.003	0.703 ± 0.016	0.721 ± 0.009	0.831 ± 0.004	N/A	N/A	**0.851 ± 0.013**	0.850 ± 0.007
	AMAZON R.	0.284 ± 0.003	0.363 ± 0.013	0.324 ± 0.013	0.358 ± 0.013	0.352 ± 0.017	0.311 ± 0.016	N/A	N/A	0.418 ± 0.061	**0.445 ± 0.016**
	MINESWEEPER	0.605 ± 0.001	0.764 ± 0.022	0.702 ± 0.092	0.713 ± 0.016	0.729 ± 0.007	0.801 ± 0.033	N/A	N/A	**0.862 ± 0.002**	0.852 ± 0.003
	TOLOKERS	0.595 ± 0.006	0.601 ± 0.034	0.628 ± 0.011	0.649 ± 0.007	0.642 ± 0.005	0.649 ± 0.022	N/A	N/A	**0.805 ± 0.003**	0.714 ± 0.011
	SQUIRREL	0.318 ± 0.010	0.318 ± 0.025	0.346 ± 0.024	0.351 ± 0.010	0.353 ± 0.013	0.338 ± 0.018	0.258 ± 0.048	0.294 ± 0.017	**0.422 ± 0.013**	0.409 ± 0.008
	CHAMELEON	0.436 ± 0.004	0.399 ± 0.020	0.414 ± 0.019	0.363 ± 0.042	0.400 ± 0.023	0.395 ± 0.021	0.378 ± 0.032	0.381 ± 0.034	0.431 ± 0.068	**0.478 ± 0.015**
INF	ER (15K)	0.227 ± 0.020	0.351 ± 0.116	0.154 ± 0.086	0.361 ± 0.003	0.364 ± 0.000	0.528 ± 0.073	OOM	0.091 ± 0.000	0.626 ± 0.003	**0.886 ± 0.011**
	ER (30K)	0.159 ± 0.020	0.189 ± 0.103	0.124 ± 0.078	0.246 ± 0.130	0.237 ± 0.121	0.442 ± 0.087	OOM	OOM	OOM	**0.760 ± 0.040**

Supplementary Table 10: Matthews correlation coefficient (MCC) for the heterophilous node-level classification tasks and two additional baselines, presented as mean $\pm$ standard deviation over 5 different runs.

	Dataset ($\uparrow$)	GraphSAGE	GT
HETEROPHILY	ROMAN EMPIRE	0.83 ± 0.00	0.84 ± 0.00
	AMAZON RATINGS	0.36 ± 0.00	0.34 ± 0.01
	MINESWEEPER	0.65 ± 0.01	0.57 ± 0.04
	TOLOKERS	0.23 ± 0.02	0.38 ± 0.01
	SQUIRREL FILTERED	0.14 ± 0.03	0.18 ± 0.01
	CHAMELEON FILTERED	0.27 ± 0.03	0.25 ± 0.04

SI 2 Additional time and memory results

(a) Training time for DOCKSTRING.

(b) Allocated memory for DOCKSTRING.

Supplementary Figure 1: The elapsed time for training for a single epoch on the DOCKSTRING dataset (in seconds), and the maximum allocated memory during this training epoch (GB).

SI 3 Sourcing and licensing

Most datasets are sourced from the PyTorch Geometric library (MIT license). Datasets with different sources include: DOCKSTRING (Apache 2.0, https://github.com/dockstring/dockstring), the heterophily datasets (MIT license, https://github.com/yandex-research/heterophilous-graphs), the peptide datasets from the Long Range Graph Benchmark project (MIT license, https://github.com/vijaydwivedi75/lrgb), the GW frontier orbital energies for QM9 (CC BY 4.0 license, https://doi.org/10.6084/m9.figshare.21610077.v1), and the Open Catalyst Project (CC BY 4.0 license for the datasets, MIT license for the Python package, https://github.com/Open-Catalyst-Project/Open-Catalyst-Dataset).

All GNN implementations used in this work are sourced from the PyTorch Geometric library (MIT license). The Graphormer and TokenGT implementations are sourced from the Huggingface project (Apache 2.0 license). The Graphormer implementation used for the 3D modelling task is sourced from the official GitHub repository (MIT license, https://github.com/microsoft/Graphormer). GraphGPS also uses the MIT license. PyTorch uses the BSD-3 license.

SI 4 Transfer learning setup

The transfer learning setup consists of randomly selected training, validation, and test sets of 25K, 5K, and respectively 10K molecules with GW calculations (from the total 133,885). This setup mimics the low amounts of high quality/fidelity data available in drug discovery and quantum simulations projects. In the transductive case, the entire dataset with DFT targets is used for pre-training (including the 10K test set compounds, but only with DFT-level measurements), while in the inductive setting the 10K set is completely excluded. Here, we perform transfer learning by pre-training a model on the DFT target for a fixed number of epochs (150) and then fine-tuning it on the subset of 25K GW calculations. In the transductive case, pre-training occurs on the full set of 133K DFT calculations, while in the inductive case the DFT test set values are removed (note that the evaluation is done on the test set GW measurements).

SI 5 Limitations

In terms of limitations, we highlight that the available libraries are not optimised for masking or custom attention patterns. This is most evident for very dense graphs (tens of thousands of edges or more). Memory efficient and Flash attention are available natively in PyTorch [46] starting from version 2.0, as well as in the xFormers library [47]. More specifically, we have tested at least 5 different implementations of ESA: (1) leveraging the `MultiheadAttention` module from PyTorch, (2) leveraging the `MultiHeadDispatch` module from xFormers, (3) a manual implementation of multihead attention, relying on PyTorch's `scaled_dot_product_attention` function, (4) a manual implementation of multihead attention, relying on xFormers' `memory_efficient_attention`, and (5) a naive implementation. Options (1) - (4) can all make use of efficient and fast implementations. However, we have observed performance differences between the 4 implementations, as well as compared to a naive implementation. This behaviour is likely due to the different low-level kernel implementations. Moreover, Flash attention does not currently support custom attention masks as there is little interest for such functionality from a language modelling perspective.

Although the masks can be computed efficiently during training, all frameworks require the last two dimensions of the input mask tensor to be of shape (L_n, L_n) for nodes or (L_e, L_e) for edges, effectively squaring the number of nodes or edges. However, the mask tensors are sparse and a sparse tensor alternative could greatly reduce the memory consumption for large and dense graphs. Such an option exists for the PyTorch native attention, but it is currently broken.

Another possible optimisation would be to use nested (ragged) tensors to represent graphs, since padding is currently necessary to ensure identical dimensions for attention. A prototype nested tensor attention is available in PyTorch; however, not all the required operations are supported and converting between normal and nested tensors is slow.

For all implementations, it is required that the mask tensor is repeated by the number of attention heads (e.g., 8 or 16). However, a notable bottleneck is encountered for the `MultiheadAttention` and `MultiHeadDispatch` variants described above, which require that the repeats happen in the batch dimension, i.e., requiring 3D mask tensors of shape $(B \times H, L, L)$, where H is the number of heads. The other two efficient implementations require a 4D mask instead, i.e., (B, H, L, L), where one can use PyTorch's `expand` function instead of `repeat`. The `expand` alternative does not use any additional memory, while `repeat` requires $\times H$ memory. Note that it is not possible to reshape the 4D tensor created using `expand` without using additional memory.

Finally, we noticed a limitation involving PyTorch's `nonzero()` tensor method, which is required as part of the edge masking algorithm (Algorithm 1). This is covered in detail in SI 9. Currently, the `nonzero()` method fails for very dense graphs. A fix would require an update to 64-bit integer limits.

SI 6 Helper functions

`consecutive` is a helper function that generates consecutive numbers starting from 0, with a length specified in its tensor argument as the difference between adjacent elements, and a second integer argument used for the last length computation, e.g., $\text{consecutive}([1, 4, 6], 10) = [0, 1, 2, 0, 1, 0, 1, 2, 3]$, and `first_unique_index` finds the first occurrence of each unique element in the tensor (sorted), e.g., $\text{first\_unique\_index}([3, 2, 3, 4, 2])$ $= [1, 0, 3]$. The implementations are available in our code base.

SI 7 Experimental setup

We follow a simple and universal experimental protocol to ensure that it is possible to compare the results of different methods and to evaluate a large number of datasets with high throughput. Below in SI 7.1 we described the grid search approach and the used search parameters. For the basic hyperparameters such as batch size and learning rate, we chose a number of reasonable hyperparameters and settings for all methods, regardless of their nature (GNN or attention-based). This includes the AdamW optimiser [68], more specifically the 8-bit version [69], learning rate (0.0001), batch size (128), mixed precision training with the `bfloat16` tensor format, early stopping with a patience of 30 epochs (100 for the very small datasets such as FREESOLV), and gradient clipping (set to the default value of 0.5). Furthermore, we used a simple learning rate scheduler that halved the learning rate if no improvement was encountered for 15 epochs (half the early stopping patience). If these parameters led to out-of-memory errors, we attempted reducing the batch size by 2 until the error was fixed, or reducing the hidden dimension as a last resort.

For Graphormer and TokenGT, we leverage the `huggingface` [70] implementation. We have adapted the TokenGT implementation to use Flash attention, which was not originally supported. For Graphormer, this optimisation is not possible since Flash attention is not compatible with some operations required by Graphormer. However, we did optimise the data loading process compared to the original `huggingface` implementation, leading to lower RAM usage.

SI 7.1 Hyper-parameter tuning

For a fair and comprehensive evaluation, we tune each algorithm for each dataset using a grid search approach and a selection of reasonable hyperparameters. These include the number of layers, the number of attention heads for GAT(v2) and graph transformers, dropout, and hidden dimensions. For graph-level GNNs, we evaluated configurations with 4 to 6 layers, hidden dimensions in $\{128, 256, 512\}$, the number of GAT heads in $\{8, 16\}$, and GAT dropout in $\{0, 0.2\}$. For node-level GNNs, we adapted the search since these tasks are more sensitive to overfitting, and used a number of layers in $\{1, 2, 4, 6\}$, hidden dimensions in $\{64, 128, 256\}$, the same GAT heads and dropout settings, and dropout after each GNN layer in $\{0, 0.2\}$. For Graphormer, TokenGT, and GraphGPS, we evaluate models with the number of layers in $\{4, 6, 8, 10\}$, the number of attention heads in $\{4, 8, 16\}$, and hidden dimensions in $\{128, 256, 512\}$. For ESA, we generally focused on models with 6 to 10 layers, with SABs at the start and end and MABs in the middle. Hidden dimensions are selected from $\{256, 512\}$ and the number of attention heads from $\{8, 16, 32\}$. We evaluate pre-LN and post-LN architectures, standard and gated MLPs, and different MLP hidden dimensions and number of MLP layers on a dataset-by-dataset basis. The best configuration is selected based on the validation loss and results are reported on the test set from 5 different runs. Based on recent reports on the performance of GNNs [30], we augmented all 6 GNN baselines with residual connections and normalisation layers for each graph convolutional layer. These strategies are not part of the original message passing specification but lead to substantial uplifts. For datasets with established splits, such as CIFAR10, MNIST, or ZINC, we use the available splits. For DOCKSTRING, only train and test splits are available, so we randomly extract 20,000 train molecules for validation. Otherwise, we generate our own splits with a train/validation/test ratio of 80%/10%/10%. All models are trained and evaluated using mixed-precision training with the `bfloat16` tensor format.

SI 7.2 Metrics

Given the scale of our evaluation, it is crucial to use an appropriate selection of performance metrics. To this end, we selected the metrics according to established and recent literature. For classification tasks, there is a growing consensus that Matthew's correlation coefficient (MCC) is the preferred metric over alternatives such as accuracy, F-score, and the area under the receiver operating characteristic curve (AUROC or ROC-AUC) [71–74]. The AUROC in particular has been shown to be problematic [75–77]. The MCC is an informative measure of a classifier's performance as it summarises the four basic rates of a confusion matrix: sensitivity, specificity, precision, and negative predictive value [75]. Similarly, the R^2 has been proven to be more informative than alternatives such as the mean absolute or squared errors for regression tasks [78]. Thus, our first choices for reporting results are the MCC and R^2, depending on the task. For comparison with leaderboard results and for specialised fields such as quantum mechanics, we also report comparable metrics (i.e., accuracy, mean absolute error, or root mean squared error).

SI 7.3 Baselines

We include classic message passing baselines in the form of GCN, GAT, and GIN due to their recent resurgence against sophisticated graph transformers and their widespread use. We also include the improved GATv2 [24] to complement GAT, and PNA for being neglected in other works despite its remarkable empirical performance. We complete the message passing baselines by including DropGNN [79], a family of provably expressive GNNs that can solve tasks beyond 1-WL in an efficient manner by randomly dropping nodes. As in the original paper, we use GIN as the main underlying mechanism and label this technique DropGIN. Regarding transformer baselines, we select Graphormer, TokenGT, and GraphGPS, not only due to their widespread use, but also due to generally outperforming previous generation graph transformers such as SAN. This selection is balanced, in the sense that Graphormer and TokenGT are part of a class of algorithms that focuses on representing the graph structure through encodings and token identifiers, while GraphGPS relies on GNNs and is thus a hybrid approach.

SI 8 Infected graph generation

The infected graphs are generated using the `InfectionDataset` from PyTorch Geometric. We generated two Erdős–Rényi (ER) graphs with different sizes:

- with 15,000 nodes, 40 infected nodes, a maximum shortest path length of 20, and an edge probability of 0.00009.

- with 30,000 nodes, 20 infected nodes, a maximum shortest path length of 20, and an edge probability of 0.00005.

These settings ensure a relatively balanced classification task for both graph sizes.

SI 9   Adaptations for Open Catalyst Project

Extending a given model to work with 3D data is not trivial, as demonstrated by the follow-up paper dedicated to extending and benchmarking Graphormer on 3D molecular problems [62]. As described in that paper, for the Open Catalyst Project (OCP) data certain pre-processing steps are taken to ensure satisfactory performance. Concretely, a set of Gaussian basis functions is used to encode atomic distances, which are not used in their raw form. The idea of encoding raw quantities to achieve expressive and orthogonal representations has also been studied by Gasteiger et al. for DimeNet [5], taking things further and using Bessel functions. This idea is prevalent in the literature and shows some of the complications of working with 3D coordinates.

In addition, OCP exhibits several unique characteristics that must be accounted for to extract the most performance out of any given model. One such property is given by periodic boundary conditions (common for crystal systems), requiring a dedicated pre-processing step. Another characteristic is the presence of 3 types of atoms: sub-surface slab atoms, surface slab atoms, and adsorbate atoms, which must be distinguished by the model. Finally, the task chosen in the benchmarking Graphormer paper is not only relaxed energy prediction, but also relaxed structure prediction, entailing the prediction of new coordinates for all atoms. This again leverages additional data in the dataset and can be considered a task with synergistic positive effects for relaxed energy prediction.

On top of this, the 3D implementation of Graphormer is unavailable on `huggingface` (the version used throughout the paper). We chose to perform experiments using a 10K training set that is provided as part of OCP, and the same validation set of size 25K. The entire OCP dataset consists of around 500K dense catalytic structures and training both Graphormer and ESA/NSA on this task would entail a computational effort larger than for any other evaluated dataset. To complicate things further, each 'graph' is dense, leading to a significant GPU memory burden, such that only small batch sizes are possible even for high-end GPUs.

Moreover, with higher batch sizes we have hit a limit of PyTorch: the `nonzero()` tensor method that is used as part of the mask computation is not defined for tensors with more elements than the 32-bit integer limit. While this operation can be chunked in smaller tensors, this induces a significant slowdown while training. Overall, this software limitation highlights the fact that current libraries are not optimised for masked attention. We present other software limitations in SI 5, as well as possible solutions, and we believe that ESA can be significantly optimised with careful software (and even hardware) design.

For the 10K train + 25K validation task, we have adapted our method to use the 3D pre-processing described above, and modified Graphormer to perform only relaxed energy prediction (i.e., without relaxed structure prediction). We used a batch size of 16 for both models, and roughly equivalent settings between NSA and Graphormer, where possible, including 4 layers, 16 attention heads, an embedding/hidden size of 256, and the same learning rate (1e-4). Both methods used mixed precision training. Here, we used the Graphormer 3D implementation from the official repository. We also note that structural information (in the form of `edge_index` tensors in PyTorch Geometric) is provided in the OCP dataset. They are derived in a similar way to PyTorch Geometric's `radius_graph()` function, which connects points based on a distance cutoff. We can use this information for ESA/NSA.

SI 10   Experimental platform

Representative versions of the software used as part of this paper include Python 3.11, PyTorch version 2.5.1 with CUDA 12.1, PyTorch Geometric 2.5.3 and 2.6.0, PyTorch Lightning 2.4.0, huggingface transformers version 4.35.2, and xFormers version 0.0.27. It is worth noting that attention masking and efficient implementations of attention are early features that are advancing quickly. This means that their behaviour might change unexpectedly and there might be bugs. For example, PyTorch 2.1.1 recently fixed a bug that concerned non-contiguous custom attention masks in the `scaled_dot_product_attention` function.

In terms of hardware, the GPUs used include an NVIDIA RTX 3090 with 24GB VRAM, NVIDIA V100 with 16GB or 32GB of VRAM, and NVIDIA A100 with 40GB and 80GB of VRAM. Recent, efficient implementations of attention are optimised for the newest GPU architectures, generally starting from Ampere (RTX 3090 and A100).

SI 11   Dataset statistics

We present a summary of all the used datasets, along with their size and the maximum number of nodes and edges encountered in a graph in the dataset (Supplementary Table 11). The last two are important as they determine the shape of the mask and of the inputs for the attention blocks. Technically, we require that the maximum number of nodes/edges is determined per batch and the tensors to be padded accordingly. This per-batch maximum is lower than the dataset maximum for most batches. However, certain operations such

as layer normalisation, if performed over the last two dimensions, require a constant value. To enable this, we use the dataset maximum.

Supplementary Table 11: Summary of used datasets, their size, and the maximum number of nodes (**N**) and edges (**E**) seen in a graph in the dataset.

	Dataset	Size	N	E
LRGB	PEPT-STRUCT	15 535	444	928
	PEPT-FUNC	15 535	444	928
NODE	PPI	24	3 480	106 754
	CORA	1	2 708	10 556
	CiteSeer	1	3 327	9 104
	ROMAN EMPIRE	1	22 662	65 854
	AMAZON RATINGS	1	24 492	186 100
	MINESWEEPER	1	10 000	78 804
	TOLOKERS	1	11 758	1 038 000
	SQUIRREL	1	2 223	93 996
	CHAMELEON	1	890	17 708
	INFECTED 15000	1	15 000	20 048
	INFECTED 30000	1	30 000	45 258
MolNet	FreeSolv	642	44	92
	LIPO	4 200	216	438
	ESOL	1 128	119	252
	BBBP	2 039	269	562
	BACE	1 513	184	376
	HIV	41 127	438	882
MOL	ZINC	249 456	38	90
	PCQM4Mv2	3 452 151	51	118
NCI	NCI1	4 110	111	238
	NCI09	4 127	111	238
CV	MNIST	70 000	75	600
	CIFAR10	60 000	150	1 200
BioInf	ENZYMES	600	126	298
	PROTEINS	1 113	620	2 098
	DD	1 178	5 748	28 534
SYNTH	SYNTHETIC	300	100	392
	SYNTHETIC NEW	300	100	396
	SYNTHIE	400	100	424
SOCIAL	IMDB-BINARY	1 000	136	2 498
	IMDB-MULTI	1 500	89	2 934
	TWITCH EGOS	127 094	52	1 572
	REDDIT THR.	203 088	97	370
	QM9	133 885	29	56
	DOCKSTRING	260 060	164	342
	MalNetTiny	5 000	4 994	20 096
	Open Catalyst Project	35 000	334	11 094