

Scaling Graph Transformers with Expander Graphs

Danica Sutherland UBC + Amii (she/her)

based on:

Exphormer: Scaling Graph Transformers with Expander Graphs (ICML 2023; arXiv:2303.06147)

Even Sparser Graph Transformers (Spexphormer; not quite on arXiv yet, but soon!)

with:



Hamed Shirzad
UBC



Ameya Velingker
Google



Balaji Venkatachalam
Google → Meta



Ali Sinop
Google
(Exphormer only)

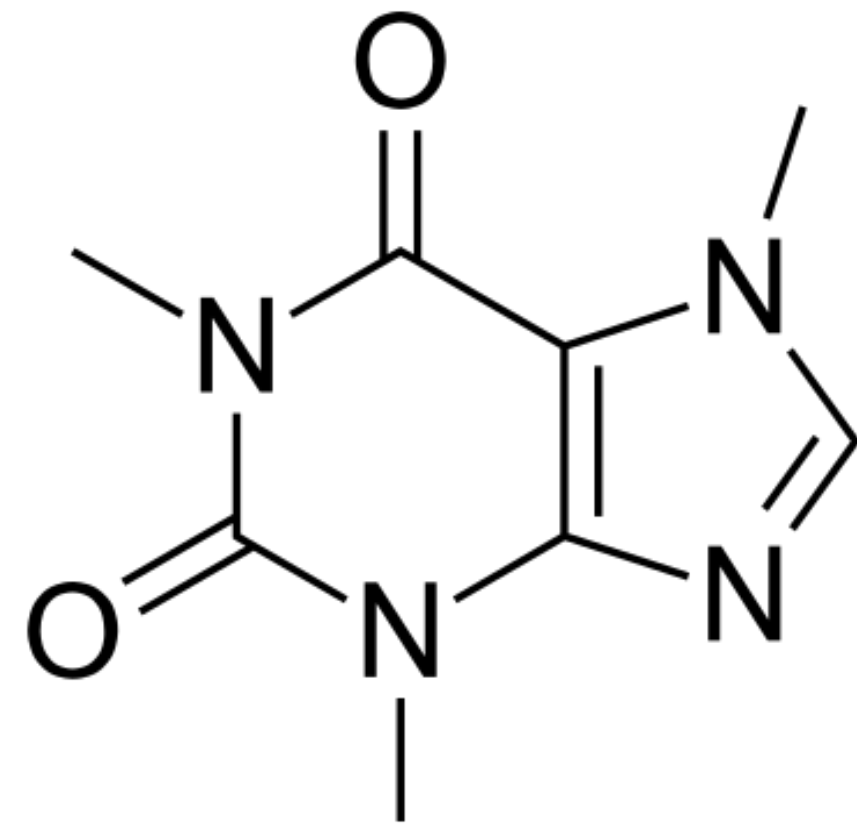


Honghao Lin
CMU
(Spexphormer only)

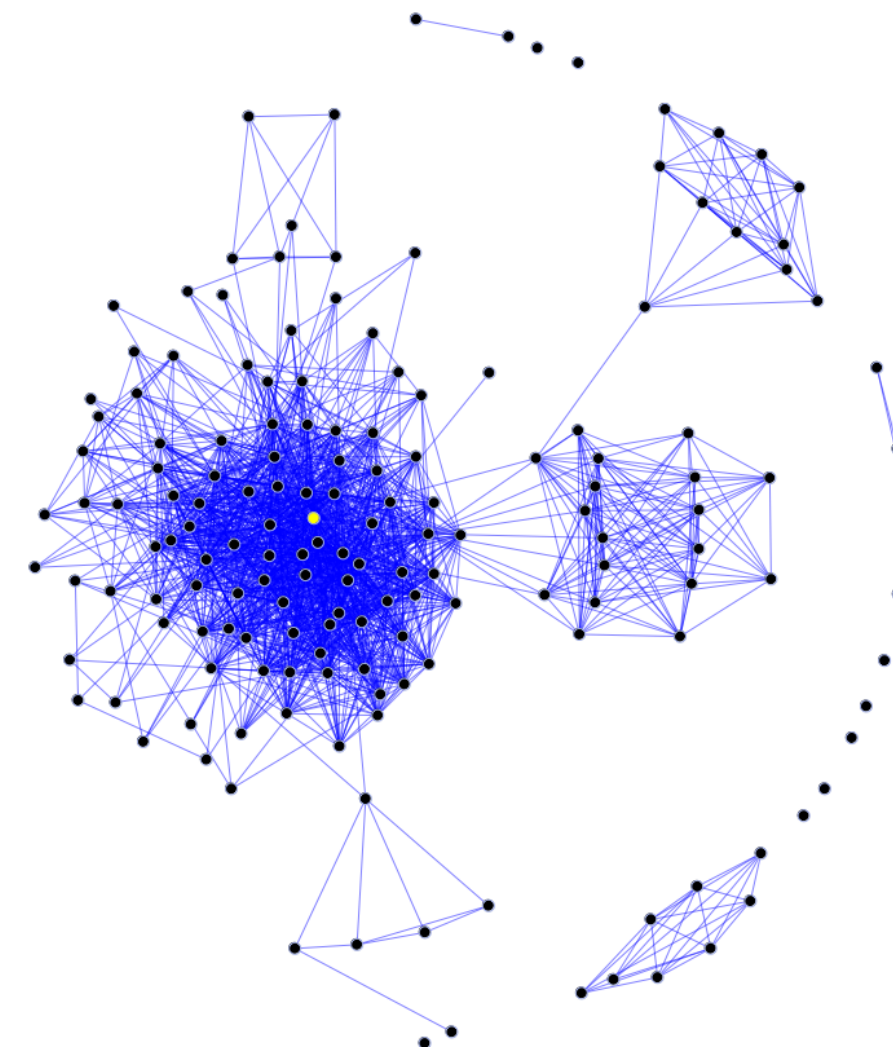


David Woodruff
CMU + Google
(Spexphormer only)

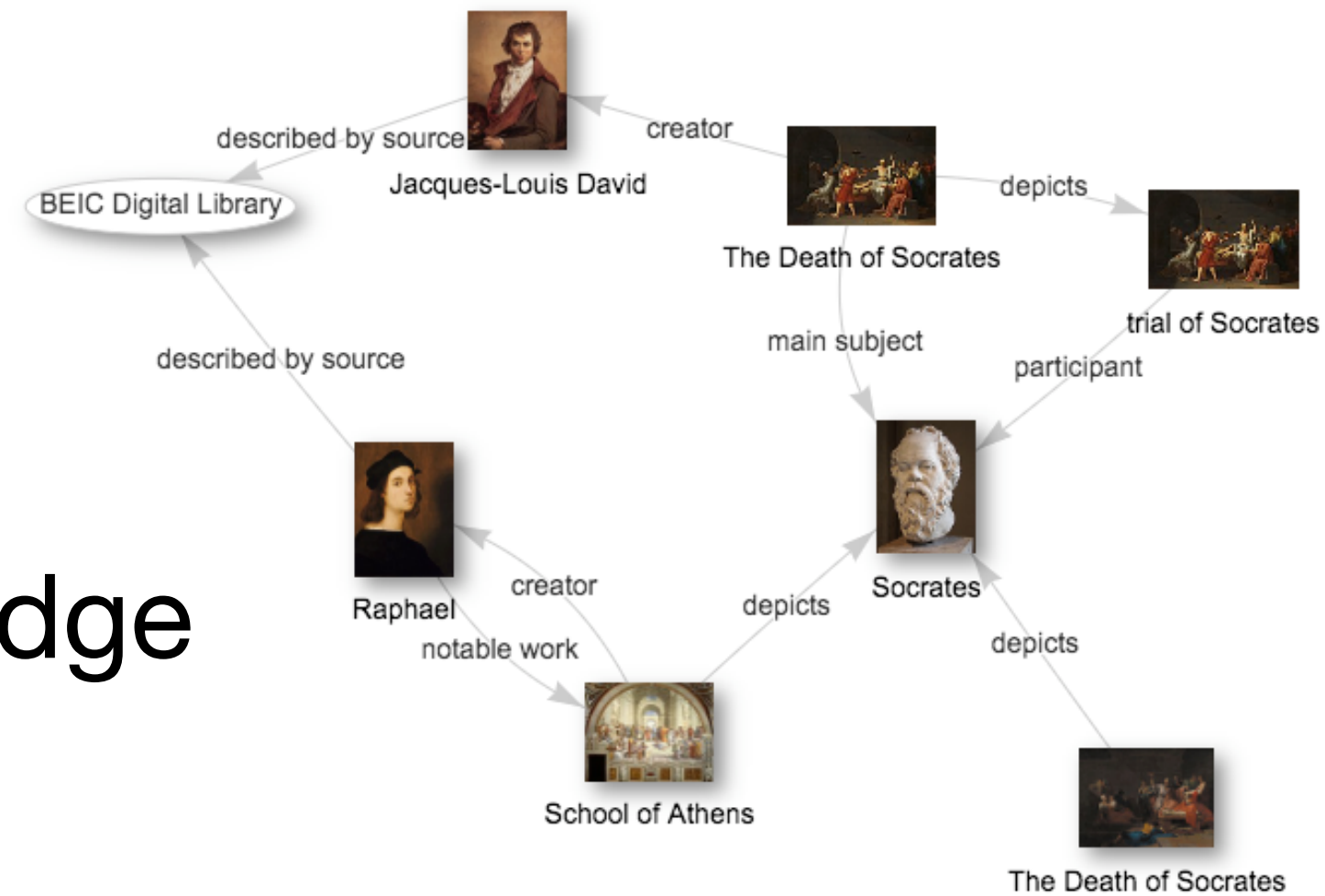
Learning on graphs



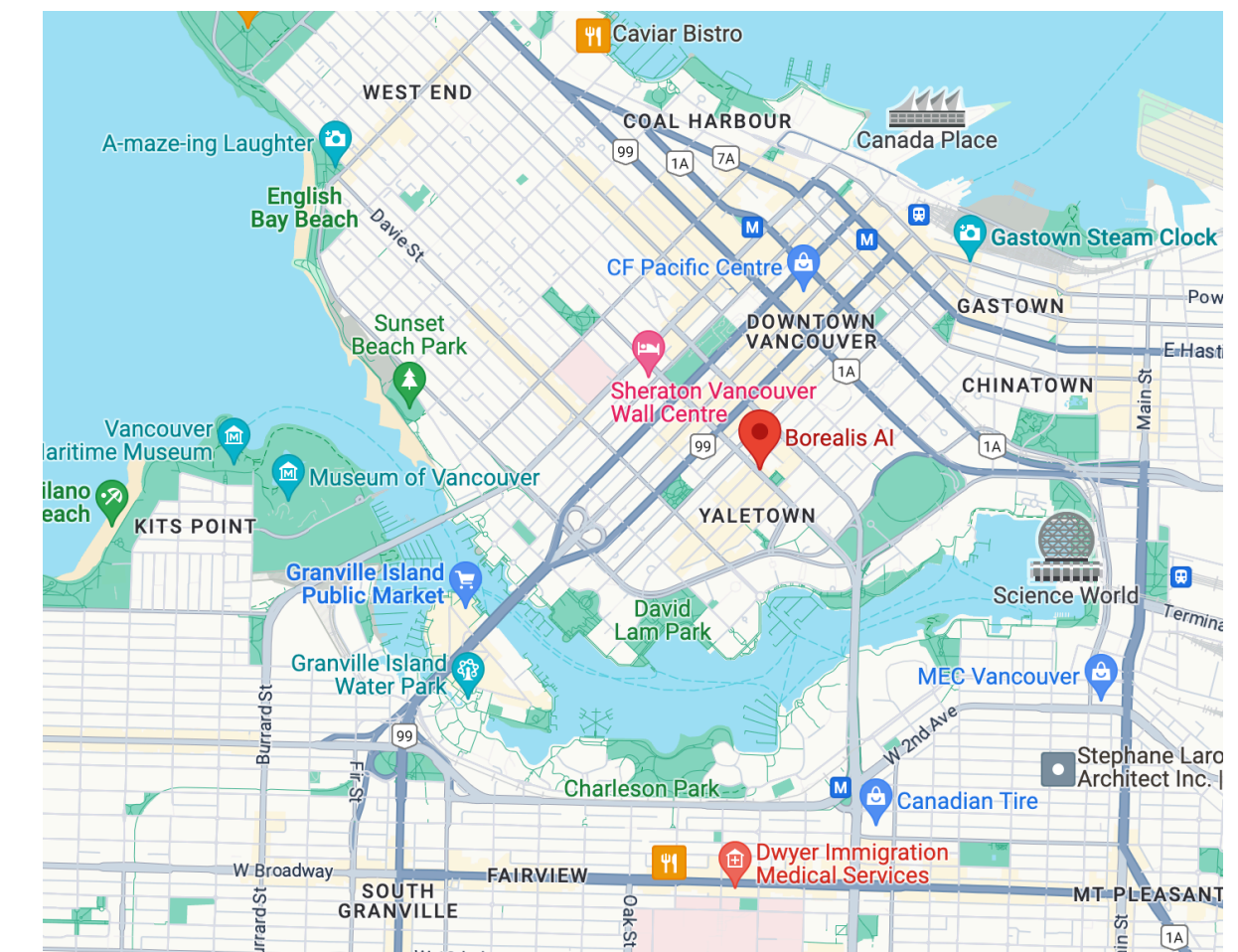
Molecular
Graphs



Social
Network
Graphs

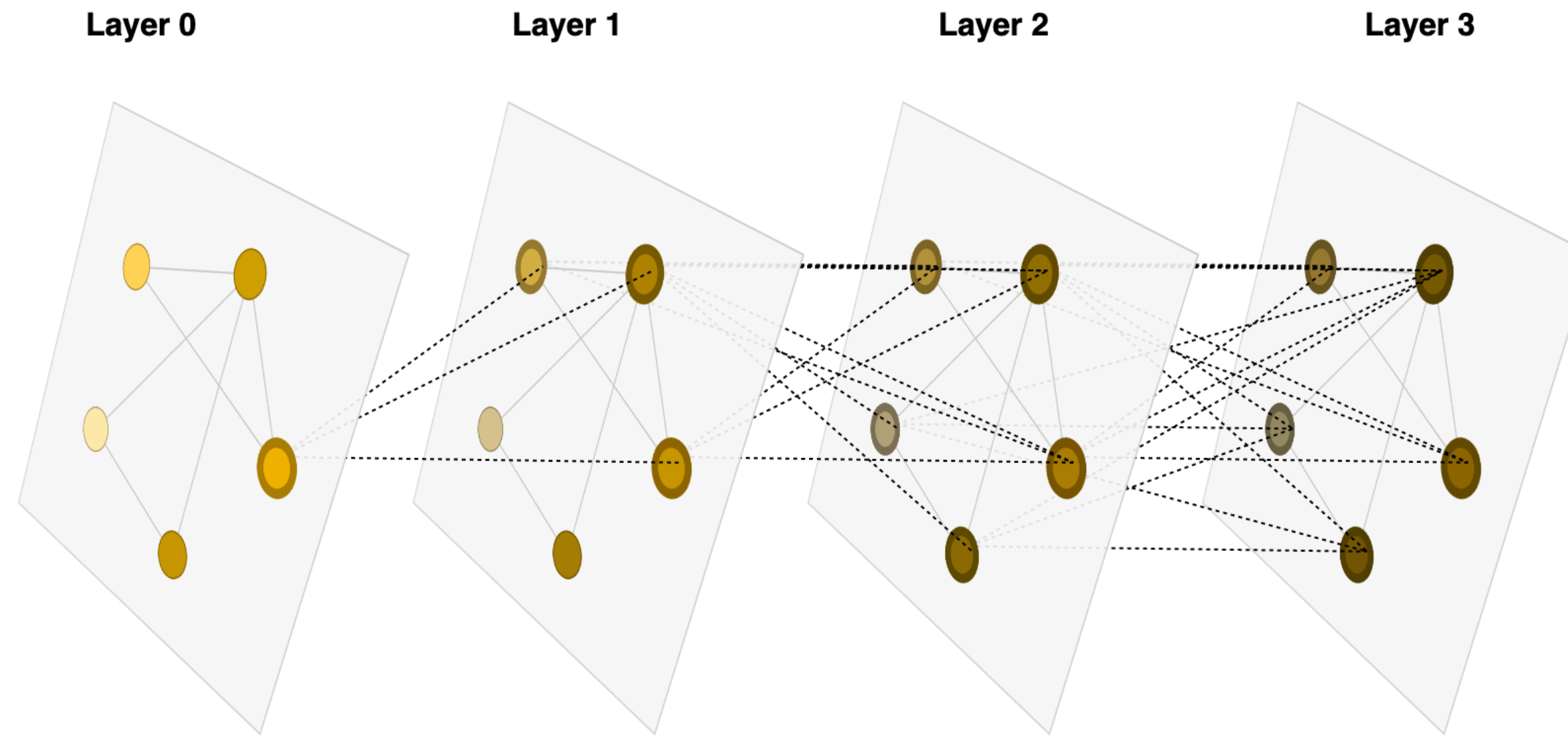


Knowledge
Graphs



Road Network Graphs

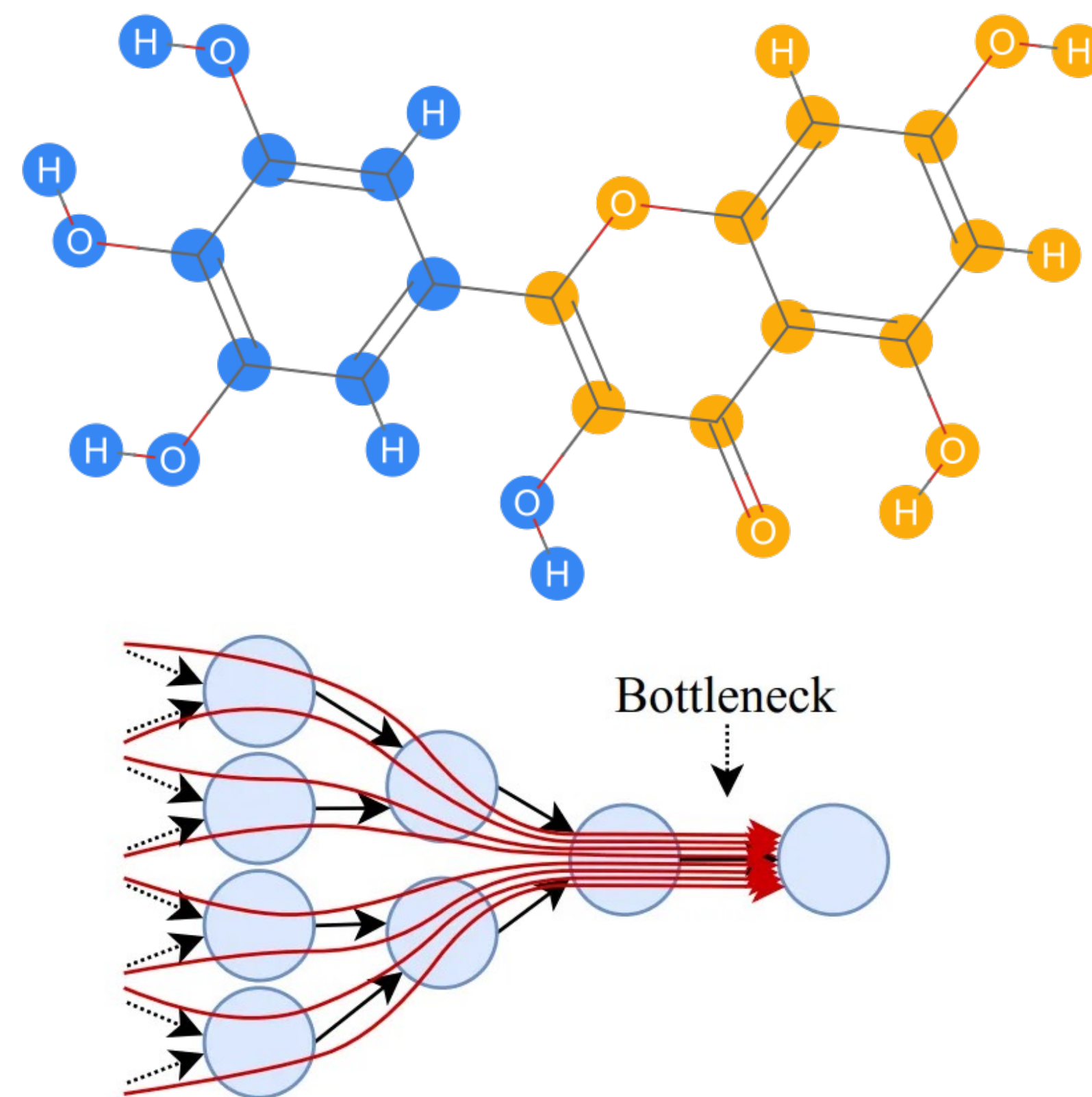
Message-passing neural networks



- **Features** at each node x_i , maybe also each edge $e_{i \rightarrow j}$
- **Message** from node j to node i : $m_{j \rightarrow i} = f_e(x_j, x_i, e_{j \rightarrow i})$ for some NN f_e
- New features $x_i^+ = f_v(x_i, \sum_j m_{j \rightarrow i})$ for some NN f_v

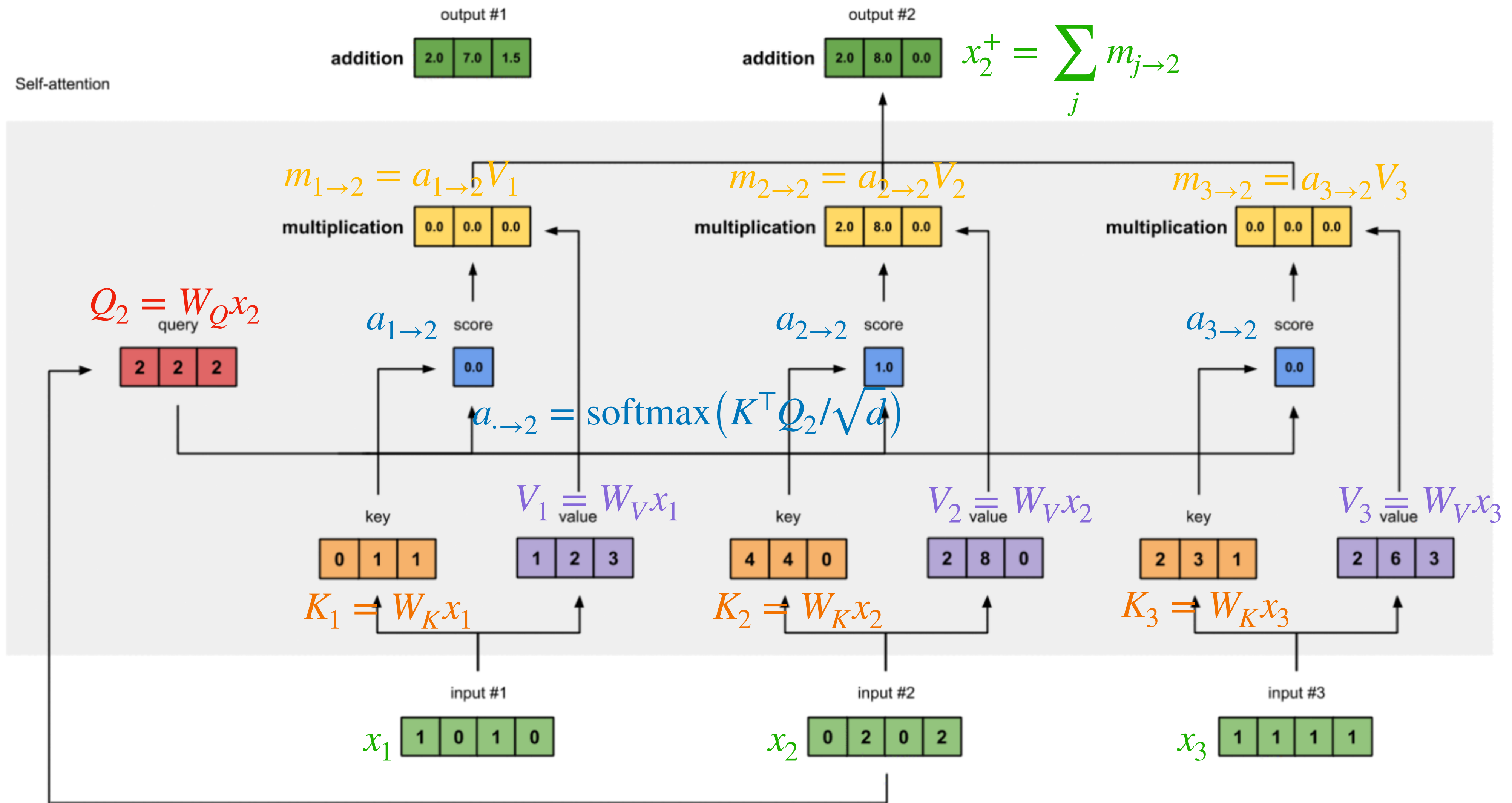
Problems with message-passing neural nets

- Information only propagates along the graph edges
 - What if there are *long-range dependencies*?
- Over-smoothing
 - In deeper layers, all node features often end up basically the same as each other
- Over-squashing



Self-attention layers

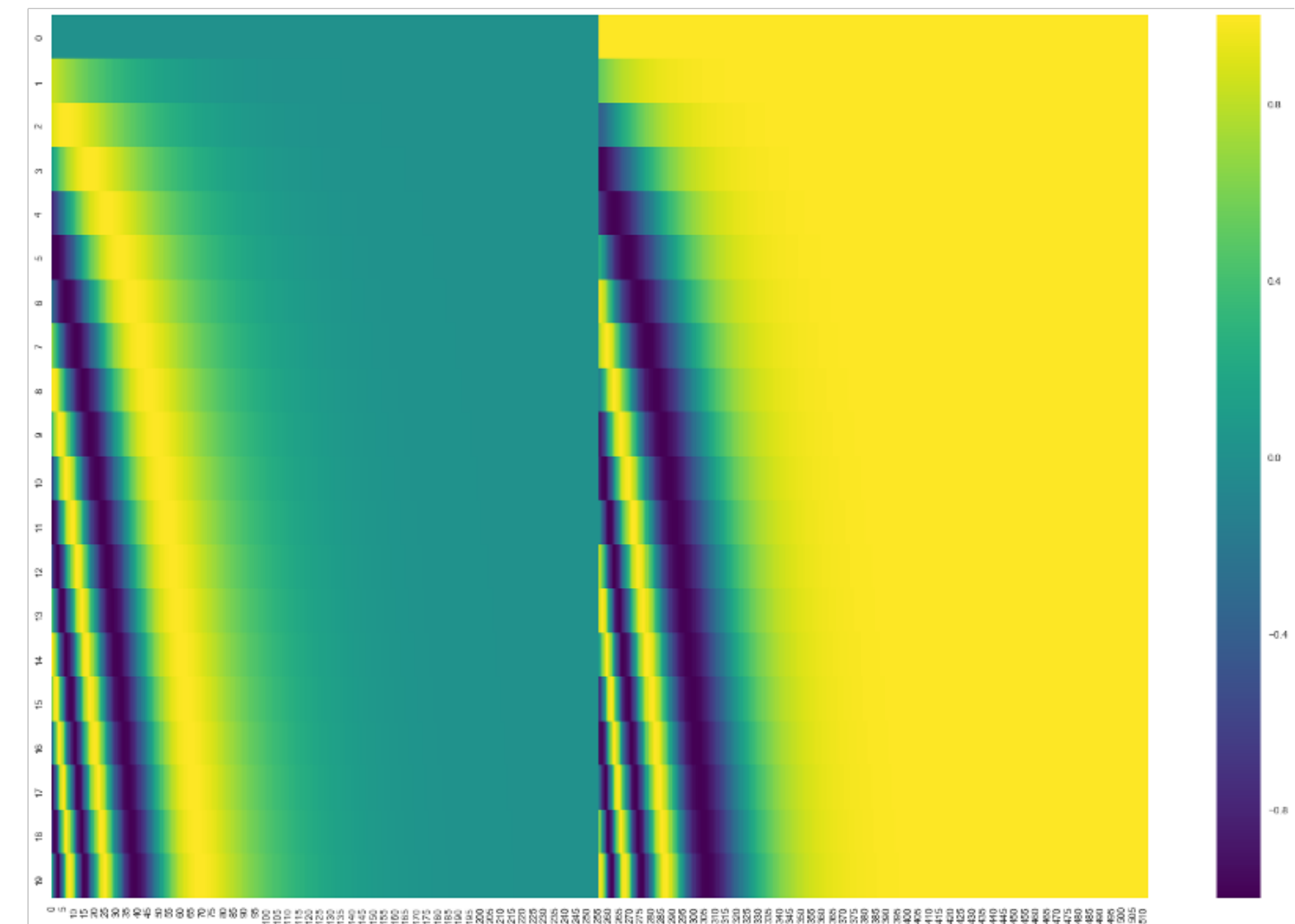
Looks like message-passing over the *full graph*



Positional encodings for sequences

- Problem: the order of the sequence is lost!
- Solution: add features for **positional encodings** that tell you “where you are”
- “Default” version: trigonometric features

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$



Positional encodings

- Trigonometric functions: **eigenfunctions of Laplacian** on Euclidean space

Eigenvalues and eigenfunctions of the Laplacian

Andrew Hassell

The setting

In this talk I will consider the Laplace operator, Δ , on various geometric spaces M .

On a Euclidean domain,

$$\Delta f = - \sum_{i=1}^n \frac{\partial^2 f}{\partial x_i^2}.$$

Examples

- The interval $[0, a]$. Eigenfunctions and eigenvalues are

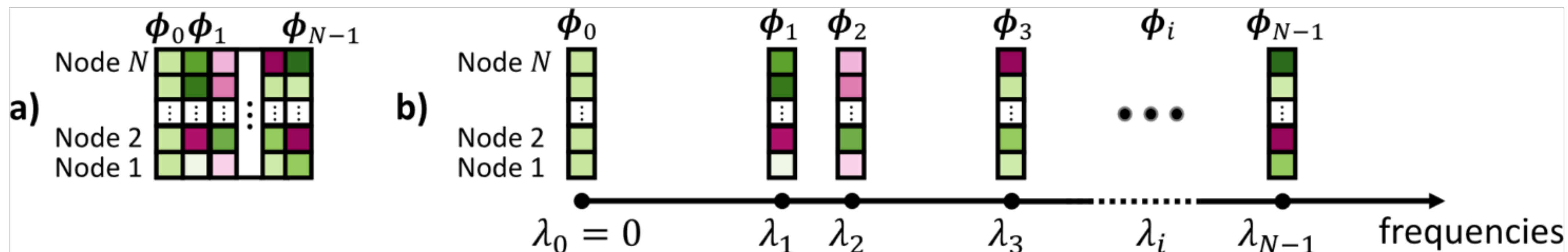
$$u_n = \sqrt{\frac{2}{a}} \sin \frac{\pi n x}{a}, \quad \lambda_n = \frac{\pi n}{a}.$$

- The torus T_π^2 . Eigenfunctions and eigenvalues are

$$u = \frac{1}{\pi} e^{ilx} e^{imy}, \quad \lambda = \sqrt{l^2 + m^2}.$$

Positional encodings

- Trigonometric functions: **eigenfunctions of Laplacian** on Euclidean space
- What should we do so that a self-attention layer “knows” about the structure of a graph?
 - “Default” choice: eigenvectors of the graph Laplacian $L = D - A$



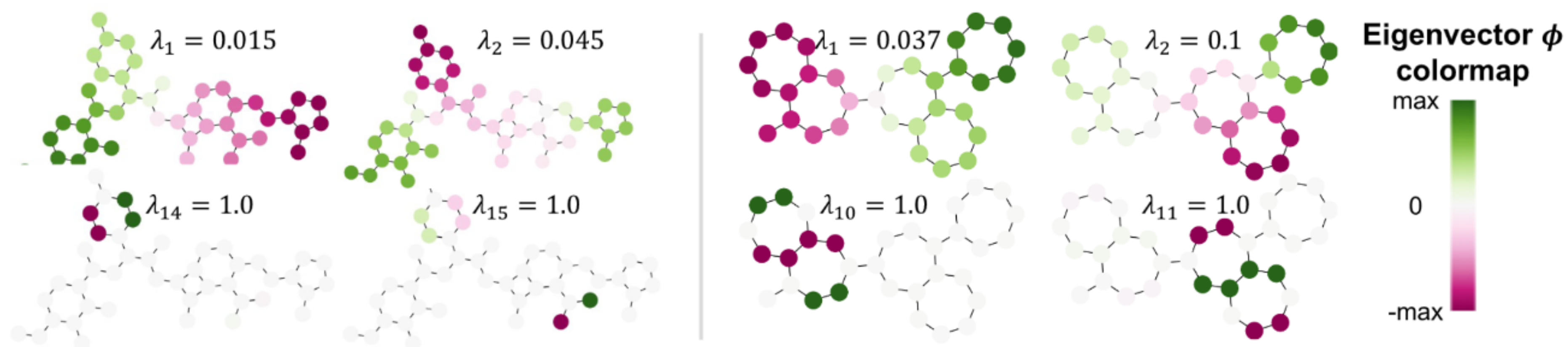
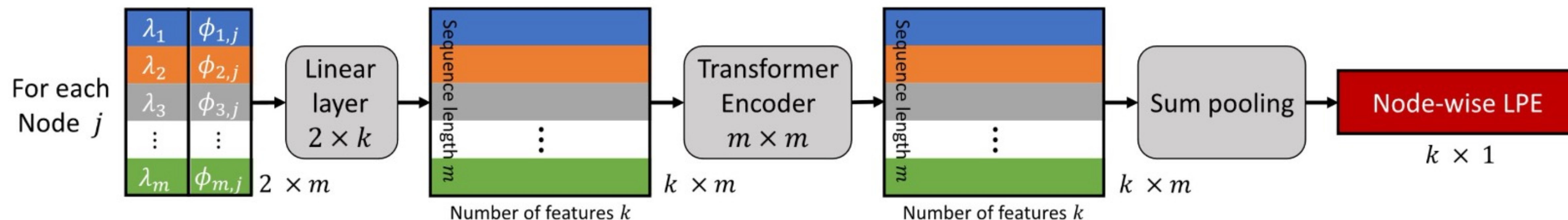


Figure 3: Examples of eigenvalues λ_i and eigenvectors ϕ_i for molecular graphs. The low-frequency eigenvectors ϕ_1, ϕ_2 are spread accross the graph, while higher frequencies, such as ϕ_{14}, ϕ_{15} for the left molecule or ϕ_{10}, ϕ_{11} for the right molecule, often resonate in local structures.

Learned positional encodings (Kreuzer et al. 2021)

- Can be helpful to post-process the positional encodings further



Spectral Attention Networks (SAN) (Kreuzer et al. 2021)

- Self-attention layers with learned positional encodings
- Intersperse with MLPs processing each node feature independently, as in Transformer encoders

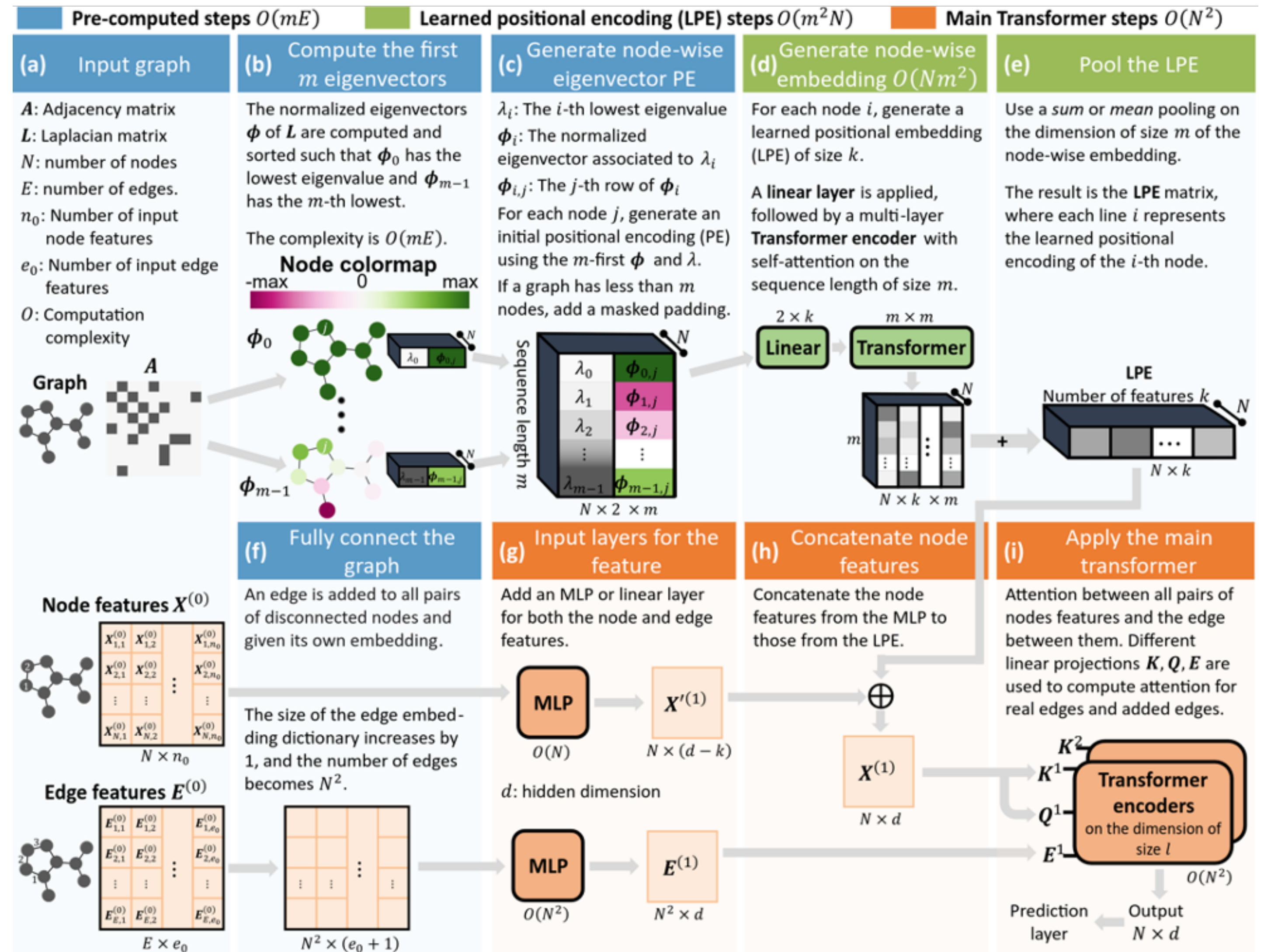


Figure 1: The proposed SAN model with the node LPE, a generalization of Transformers to graphs.

Full attention and position encodings help

Model details		ZINC	PATTERN	CLUSTER	MOLHIV
Attention	LPE	MAE	% ACC	% ACC	% ROC-AUC
Sparse	-	0.267 ± 0.032	83.613 ± 0.663	75.683 ± 0.098	73.46 ± 0.71
Sparse	Node	0.198 ± 0.004	81.329 ± 2.150	75.738 ± 0.106	76.61 ± 0.62
Full	-	0.392 ± 0.055	86.322 ± 0.049	76.447 ± 0.177	73.84 ± 1.80
Full	Node	0.157 ± 0.006	86.441 ± 0.040	76.691 ± 0.247	77.57 ± 0.61




Figure 6: Ablation study on datasets from [15, 21] for the node LPE and full graph attention, with no hyperparameter tuning other than γ taken from Figure 5. For a given dataset, all models use the same hyperparameters, but the hidden dimensions are adjusted to have $\sim 500k$ learnable parameters. Means and uncertainties are derived from four runs, with different seeds (except MolHIV).

Message passing vs graph transformers

Message passing networks

Update across edges of input graph

✓ Capture inductive bias from input graph topology

✓ Efficient computation: $O(N + M)$

✗ Difficulty with long-range dependencies

✗ Oversmoothing, oversquashing

Positional and structural encodings

Our work!

Graph transformers

Use global attention

✓ Computation graph can be different from input graph

✓ Long-range modeling

✓ Universal Approximation Theorem

✗ Knowledge of graph structure

✗ Loss of inductive bias from graph

✗ Inefficient computation: $O(N^2)$

GraphGPS

(Rampášek et al., 2022)

- Framework to combine transformer layers with message-passing layers
- MPNN layers better at preserving graph structure
- Transformer layers better at certain kinds of dependence

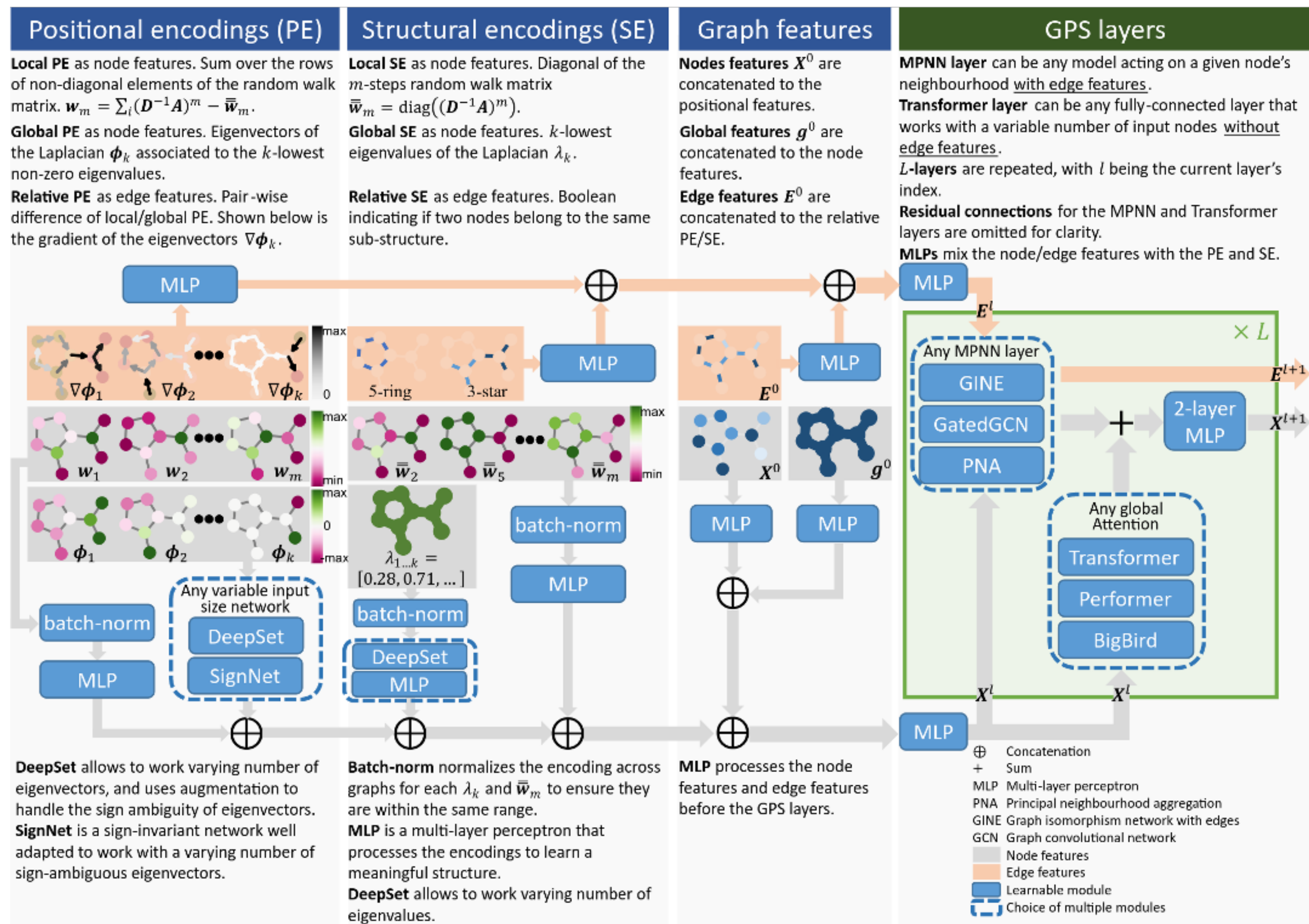
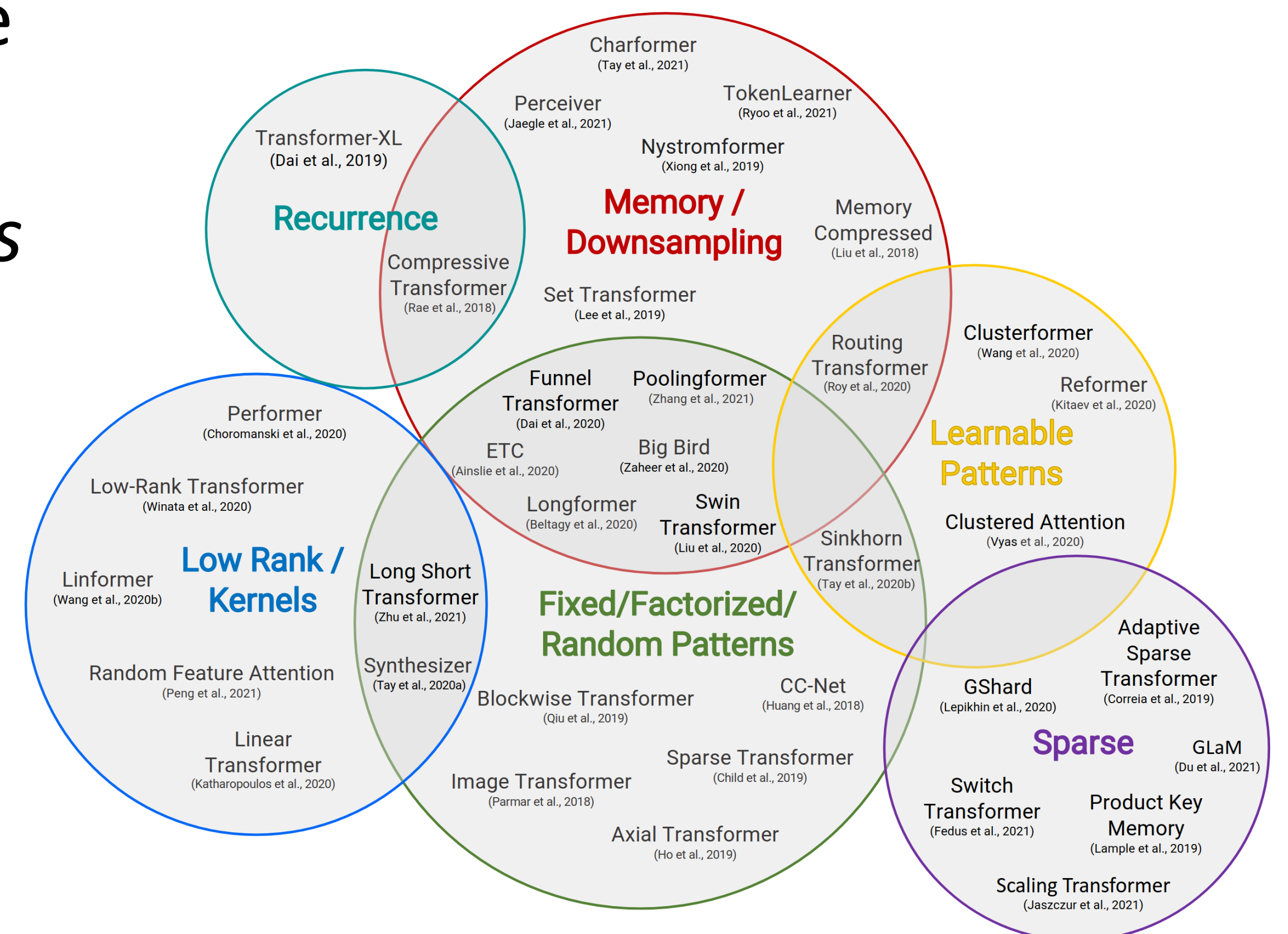


Figure 1: Modular GPS graph Transformer, with examples of PE and SE. Task specific layers for node/graph/edge-level predictions, such as pooling or output MLP, are omitted for simplicity.

Sparse attention and Exphormer

Reducing the N^2 attention cost

- Lots of work on more-efficient attention mechanisms for sequences
- GraphGPS paper tried to apply some to graphs...but it hurt accuracy a lot
- Need something *designed for graphs*

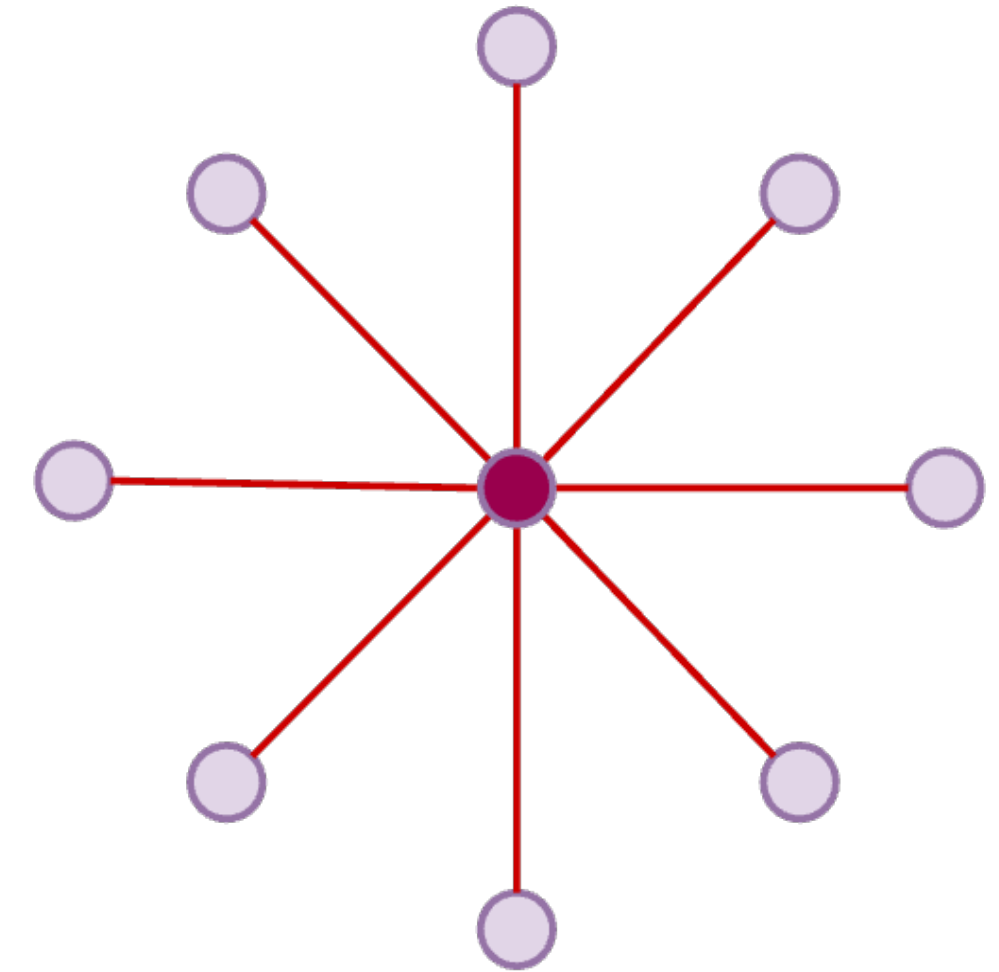


Attention graph

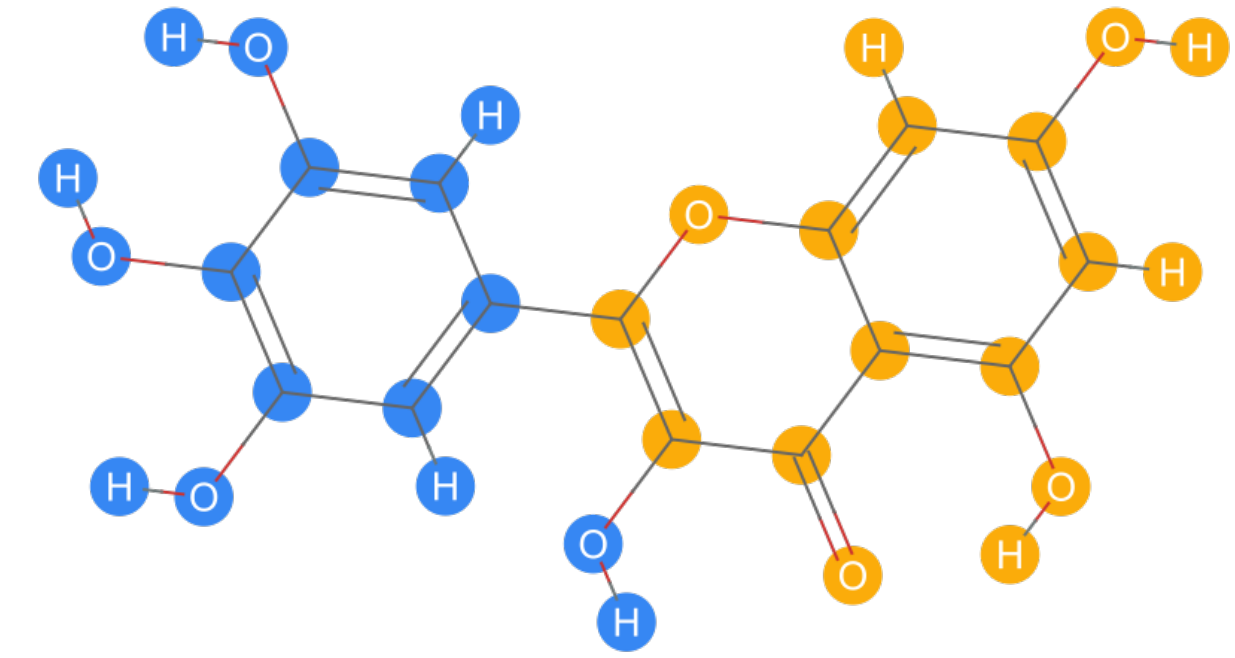
- We'll take an approach of “sparse attention”
- Start by “running” attention along the graph edges (both ways)
 - Adjacent nodes are reasonably likely to “matter” to each other
- Full transformers **augment** this attention graph with the complete graph
 - Every node connected to every other
 - Allows easy long-range communication
 - Adds **too many edges!** Makes it slow + memory-hungry

Augmenting attention with virtual nodes

- Add a “**virtual**” node (or four) that’s connected to *every* graph node
- Now any node can “talk to” any other node in at most 2 hops
- Also common in MPNNs
- Problem: **oversquashing**



Expander graphs



- How can we connect far-away parts of the graph *without* severe oversquashing?
- We want a **sparse approximation** to the complete graph
 - No too many edges
 - Can get from anywhere to anywhere quickly
 - Have many routes from place to place, to avoid oversquashing
- Mathematical concept called **expander graphs**
 - Approximate the complete graph (in many senses), with low degree

Expander graphs

Theorem 4.1. *A d -regular ϵ -expander G on n vertices spectrally approximates the complete graph K_n on n vertices:²*

$$(1 - \epsilon) \frac{1}{n} L_K \preceq \frac{1}{d} L_G \preceq (1 + \epsilon) \frac{1}{n} L_K.$$

Spectral approximation is known to preserve the cut structure in graphs. As a result, a sparse attention mechanism based on expander edges retains spectral properties of the full attention mechanism: cuts, vertex expansion, and so on.

Lemma 4.2. *(Alon, 1986) Let $G = (V, E)$ be a d -regular ϵ -expander graph on $n = |V|$ nodes. For any initial distribution $\pi^{(0)} : V \rightarrow \mathbb{R}^+$ and any $\delta > 0$, $\pi^{(t)}$ satisfies*

$$\|\pi^{(t)} - \frac{1}{n}\|_1 \leq \delta$$

as long as $t = \Omega\left(\frac{1}{\epsilon} \log(n/\delta)\right)$.

Theorem 4.3. *(Alon, 1986) Suppose $G = (V, E)$ is a d -regular ϵ -expander graph on n vertices. Then, for every vertex v and $k \geq 0$, the k -hop neighborhood $B(v, k) = \{w \in V : d(v, w) \leq k\}$ has*

$$|B(v, k)| \geq \min\{(1 + c)^k, n\}$$

for some constant $c > 0$ depending on d, ϵ . In particular, we have that $\text{diam}(G) = O_{d, \epsilon}(\log n)$.

Corollary 4.4. *If a sparse attention mechanism on n nodes is a d -regular ϵ -expander graph, then stacking $O_{d, \epsilon}(\log n)$ transformer layers models all pairwise node interactions.*

Expander graphs

CS366: Graph Partitioning and Expanders

[\[general info\]](#) [\[lecture notes\]](#) [\[exams and projects\]](#)

what's new

- 2/14 [midterm](#)

general information

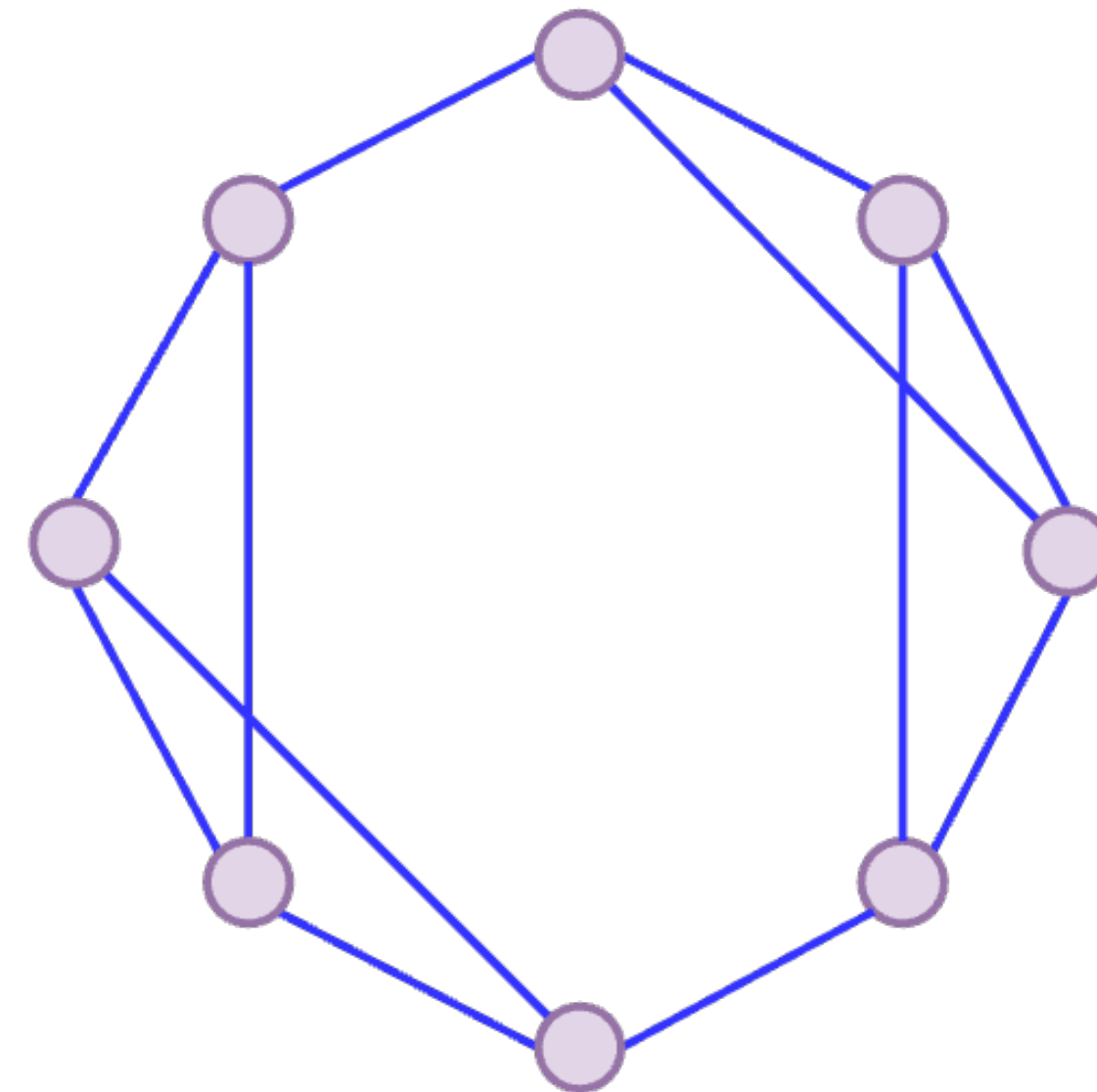
Instructor: [Luca Trevisan](#), Gates 474, Tel. 650 723-8879, email *trevisan at stanford dot edu*

Expander graphs turn out to have a slew of applications not only in math but also in computer science and physics. They can be used to create error-correcting codes and to figure out when simulations based on random numbers converge to the reality they are trying to simulate. Neurons can be modeled in a graph that some researchers believe forms an expander, due to the limited space for connections inside the brain. The graphs are also useful to geometers who try to understand how to traverse complicated surfaces, among other problems.

<https://www.quantamagazine.org/new-proof-shows-that-expander-graphs-synchronize-20230724/>

Constructing expander graphs

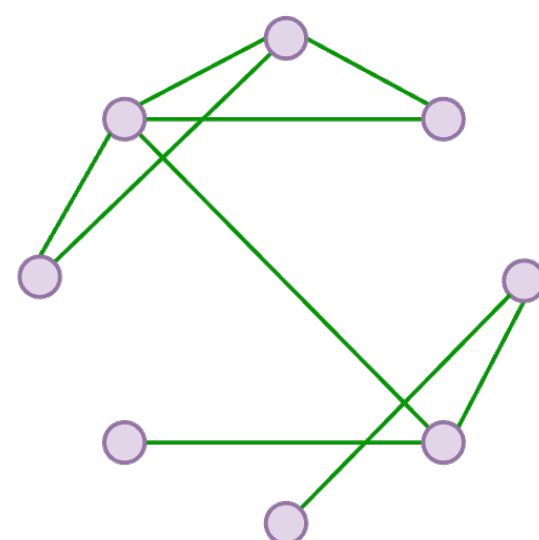
- How to make an expander graph with max degree $\leq d$:
 - Sample $d/2$ random Hamiltonian cycles, combine these as edges
 - Check (from the eigenvalues of the Laplacian) if it's an expander
 - If not, try again (happens rarely)



Components of Exphormer

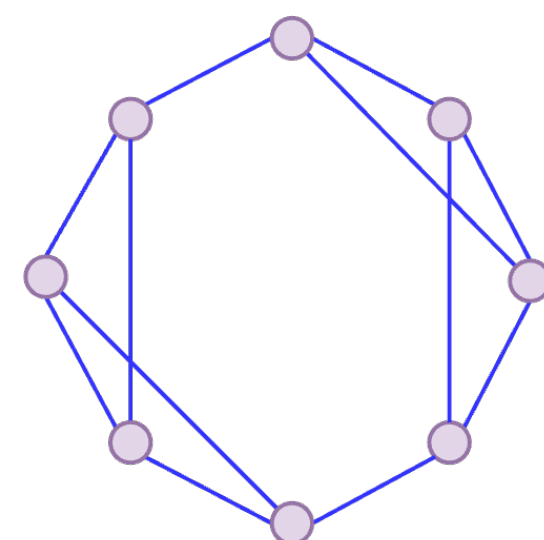
Original graph

- Preserve locality from the original graph



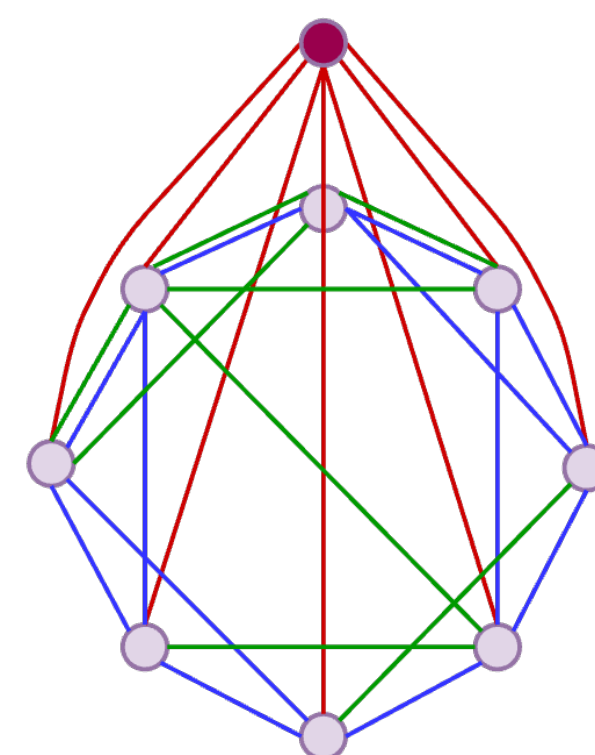
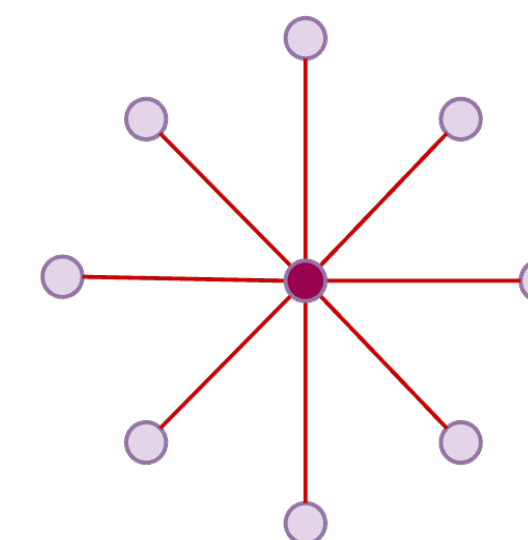
Expander graph

- Random walk mixing
- Constant degree
- $O(N)$ edges



Virtual node(s)

- "Storage sink"
- Short connections between all node pairs



Exphormer: Combine all three to form the interaction graph

Use edge features to let the network know which kind of edge it is

Universal approximation

Theorem E.3. *Suppose H is the attention graph of EXPHORMER (which contains n graph nodes and potentially more virtual nodes), augmented with self loops on all nodes. Suppose H satisfies at least one of the following:*

- 1. H contains at least one node which is connected to all n graph nodes (i.e., at least one virtual node is included).*
- 2. The underlying expander graph of H contains a Hamiltonian path.*

Then, it follows that a sparse transformer model, with positional encodings and an attention mechanism following H , can universally approximate continuous functions $f : [0, 1]^{d \times n} \rightarrow \mathbb{R}^{d \times n}$. That is, for any $1 < p < \infty$ and $\epsilon > 0$, there exists a sparse transformer network g , which uses the attention graph H and some positional encodings, such that $\ell^p(f, g) < \epsilon$.

Experimental Results: Various Datasets

Table 1. Comparison of EXPHORMER with baselines on various datasets. Best results are colored in **first**, **second**, **third**.

Model	CIFAR10 Accuracy \uparrow	MalNet-Tiny Accuracy \uparrow	MNIST Accuracy \uparrow	CLUSTER Accuracy \uparrow	PATTERN Accuracy \uparrow
GCN (Kipf & Welling, 2017)	55.71 \pm 0.381	81.0	90.71 \pm 0.218	68.50 \pm 0.976	71.89 \pm 0.334
GIN (Xu et al., 2018)	55.26 \pm 1.527	88.98 \pm 0.557	96.49 \pm 0.252	64.72 \pm 1.553	85.39 \pm 0.136
GAT (Veličković et al., 2018)	64.22 \pm 0.455	92.1 \pm 0.242	95.54 \pm 0.205	70.59 \pm 0.447	78.27 \pm 0.186
GatedGCN (Bresson & Laurent, 2017; Dwivedi et al., 2020)	67.31 \pm 0.311	92.23\pm0.65	97.34 \pm 0.143	73.84 \pm 0.326	85.57 \pm 0.088
PNA (Corso et al., 2020)	70.35 \pm 0.63	–	97.94 \pm 0.12	–	–
DGN (Beaini et al., 2021)	72.84\pm0.417	–	–	–	86.68 \pm 0.034
CRaWI (Toenshoff et al., 2021)	69.01 \pm 0.259	–	97.94 \pm 0.050	–	–
GIN-AK+ (Zhao et al., 2022b)	72.19 \pm 0.13	–	–	–	86.85\pm0.057
SAN (Kreuzer et al., 2021)	–	–	–	76.69 \pm 0.65	86.58 \pm 0.037
K-Subgraph SAT (Chen et al., 2022a)	–	–	–	77.86 \pm 0.104	86.85\pm0.037
EGT (Hussain et al., 2021)	68.70 \pm 0.409	–	98.17\pm0.087	79.23\pm0.348	86.82\pm0.020
GraphGPS (Rampásek et al., 2022)	72.30\pm0.356	93.50\pm0.41	98.05\pm0.126	78.02\pm0.180	86.69 \pm 0.059
EXPHORMER (ours)	74.69\pm0.125	94.02 \pm 0.209	98.55 \pm 0.039	78.07 \pm 0.037	86.74 \pm 0.015

Experimental Results: Long-Range Benchmark

Table 3. Comparison of EXPHORMER with baselines from the Long-Range Graph Benchmarks (LRGB, Dwivedi et al., 2022). Best results are colored in **first**, **second**, **third**.

Model	PascalVOC-SP F1 score \uparrow	COCO-SP F1 score \uparrow	Peptides-Func AP \uparrow	Peptides-Struct MAE \downarrow	PCQM-Contact MRR \uparrow
GCN	0.1268 \pm 0.0060	0.0841 \pm 0.0010	0.5930 \pm 0.0023	0.3496 \pm 0.0013	0.3234 \pm 0.0006
GINE	0.1265 \pm 0.0076	0.1339 \pm 0.0044	0.5498 \pm 0.0079	0.3547 \pm 0.0045	0.3180 \pm 0.0027
GatedGCN	0.2873 \pm 0.0219	0.2641 \pm 0.0045	0.5864 \pm 0.0077	0.3420 \pm 0.0013	0.3218 \pm 0.0011
GatedGCN+RWSE	0.2860 \pm 0.0085	0.2574 \pm 0.0034	0.6069 \pm 0.0035	0.3357 \pm 0.0006	0.3242 \pm 0.0008
Transformer+LapPE	0.2694 \pm 0.0098	0.2618 \pm 0.0031	0.6326 \pm 0.0126	0.2529 \pm 0.0016	0.3174 \pm 0.0020
SAN+LapPE	0.3230 \pm 0.0039	0.2592 \pm 0.0158*	0.6384 \pm 0.0121	0.2683 \pm 0.0043	0.3350 \pm 0.0003
SAN+RWSE	0.3216 \pm 0.0027	0.2434 \pm 0.0156*	0.6439 \pm 0.0075	0.2545 \pm 0.0012	0.3341 \pm 0.0006
GraphGPS	0.3748 \pm 0.0109	0.3412 \pm 0.0044	0.6535 \pm 0.0041	0.2500 \pm 0.0005	0.3337 \pm 0.0006
Exphormer (ours)	0.3975 \pm 0.0037	0.3455 \pm 0.0009	0.6527 \pm 0.0043	0.2481 \pm 0.0007	0.3637 \pm 0.0020

Experimental Results: Larger Graphs

Table 4. Accuracy of models with different attention mechanisms on transductive graph datasets (numbers in top rows, other than arXiv, are from [Chen et al., 2022b](#)). [Chen et al.](#) did not report NAGphormer results on this dataset.

Model	ogbn-arxiv	Computer	Photo	CS	Physics
SAN	OOM	89.83 ± 0.16	94.86 ± 0.10	94.51 ± 0.15	OOM
GraphGPS	OOM	OOM	95.06 ± 0.13	93.93 ± 0.12	OOM
NAGphormer	NA	91.22 ± 0.14	95.49 ± 0.11	95.75 ± 0.09	97.34 ± 0.03
EXPHORMER	72.44 ± 0.28	91.59 ± 0.31	95.27 ± 0.42	95.77 ± 0.15	97.16 ± 0.13

What if the original graph is too big?

Memory usage

- GPUs only have so much memory available
- Exphormer only adds $\mathcal{O}(N)$ extra edges to the original graph
- But what if the original graph already had too many edges to run on?
 - ogbn-proteins has 132,000 nodes (fine) but almost 40,000,000 edges!
- Can easily run out of GPU memory even with just original-graph attention

Which edges matter?

- Attention on graphs tends to be kind of sparse, ish
- If we could throw away unimportant edges beforehand, we could afford to do our optimization on the rest
- But how to tell which edges matter if we can't afford to train?
- Idea: train **another network** to tell us which edges matter

Attention score estimation

- Idea I: narrow nets are much cheaper in memory than wide nets (lower-dimensional hidden features)
- Idea II: CPU memory is much cheaper than GPU memory

Node characteristics

nodes ▲	cores ⇅	available memory ⇅	CPU ⇅	storage ⇅	GPU ⇅
3	64	4000G or 4096000M	2 x AMD Rome 7502 @ 2.50 GHz 128M cache L3	1 x 960G SSD	-
33	64	2009G or 2057500M	2 x AMD Rome 7532 @ 2.40 GHz 256M cache L3	1 x 960G SSD	-
159	48	498G or 510000M	2 x AMD Milan 7413 @ 2.65 GHz 128M cache L3	1 x SSD of 3.84 TB	4 x NVidia A100SXM4 (40 GB memory), connected via NVLink

- Train a **narrow** network **on CPU** to tell which edges matter

Attention scores agree across widths

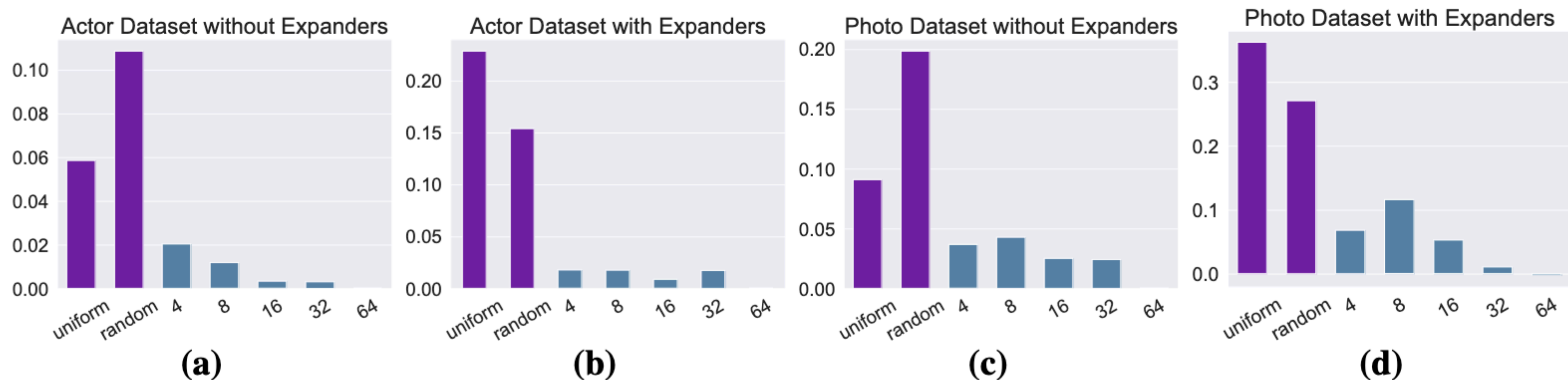


Figure 2: Energy distance between the attention scores of various networks to a network of width 64. “Uniform” refers to the baseline placing uniform scores on each neighbor, while “random” refers to the baseline with uniformly distributed logits. The remaining bars refer to networks trained on the appropriately labeled width.

Caveat I: adding layer normalization to value matrices, to make attention scores more comparable

Caveat II: using an annealed temperature schedule to encourage sparser attention

Theory

- Have some partial results about existence of narrow transformers approximating wide ones
- The first attention layer can be well-approximated with only $\log(N)$ width
- Following MLP + attention layers...not totally sure 😅

Choosing the important edges

- Once we know attention scores, how do we decide which to keep?
 - Sample k edges for each node proportionally to the attention score
 - Do different samples per graph, and resample in each epoch
- Theory: approximates attention matrix in spectral norm well
(extending proof of Achlioptas, Karnin, Liberty, NeurIPS 2013)
- Using **reservoir sampling** makes it run much faster than default implementation
- Once we have the sparsified graph,
run a wide transformer (on GPU) on only that sparsified graph

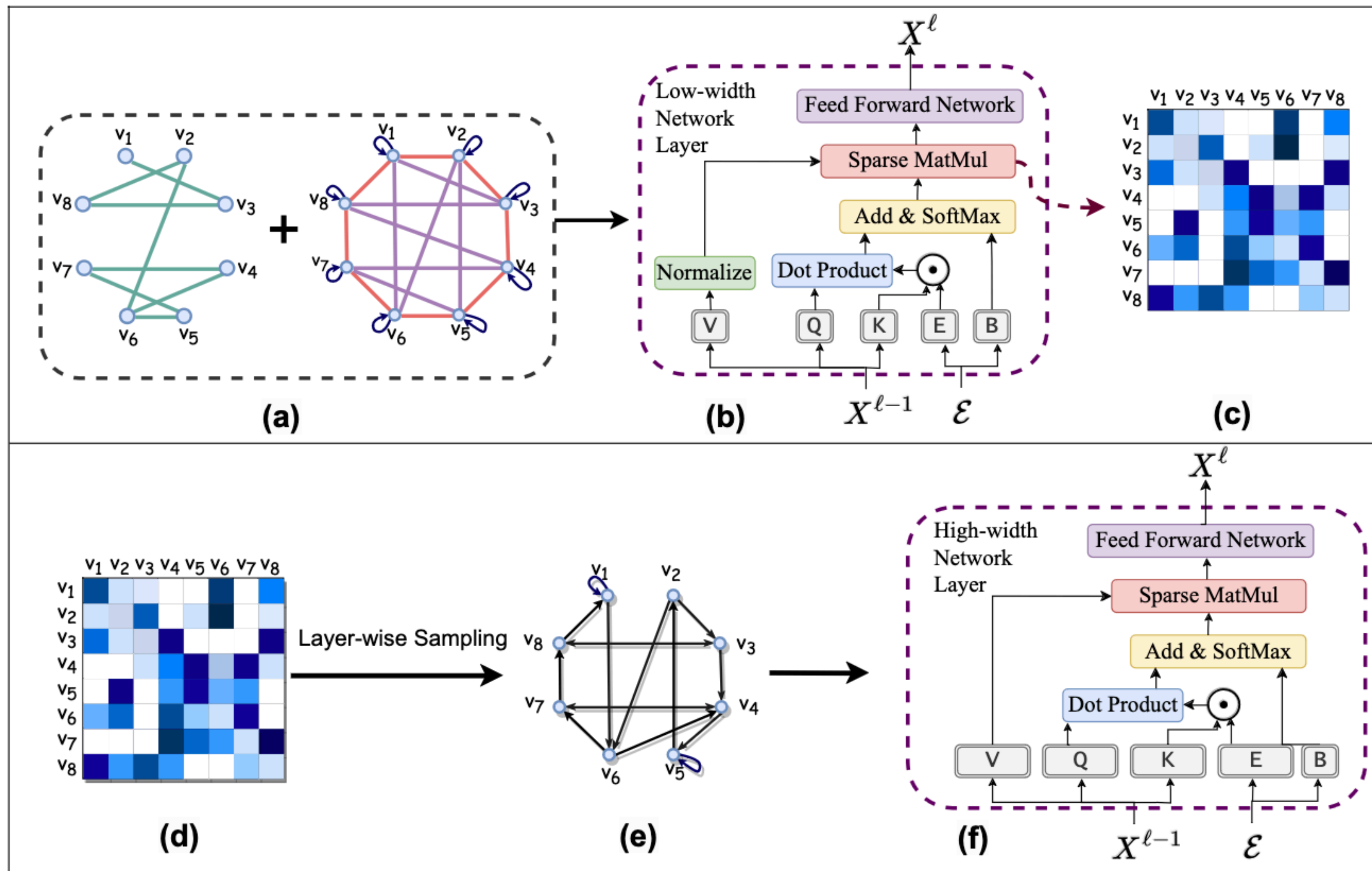


Figure 1: Steps of our method. (a) The attention mechanism for the attention score estimator network combines graph edges with an expander graph and self-loops. The expander graphs are constructed by combining a small number of Hamiltonian cycles – here two, in red and in purple – then confirming the spectral gap is large enough. (b) Self-attention layers in the estimator network use this sparse attention mechanism; its self-attention layers normalize \mathbf{V} . (c, d) Attention scores are extracted from this network for each layer, and used to sample, in (e), a sparse directed graph, which becomes the attention graph for the final network (f). This network, with a much larger feature dimension, does not normalize \mathbf{V} .

Advantage: regular graph

- Irregularly-shaped matrix multiplications are **slow on GPU**
- If each node has exactly k neighbours, we instead can do batched matrix multiplication with consistent shapes (much faster!)

Advantage: neighbourhood sampling

- Can do minibatching by
 - Start with some “core nodes”
 - Include their entire “receptive field” of other nodes that affect them
- Compared to other minibatching strategies on graphs, guarantees that minibatching doesn't bias the process

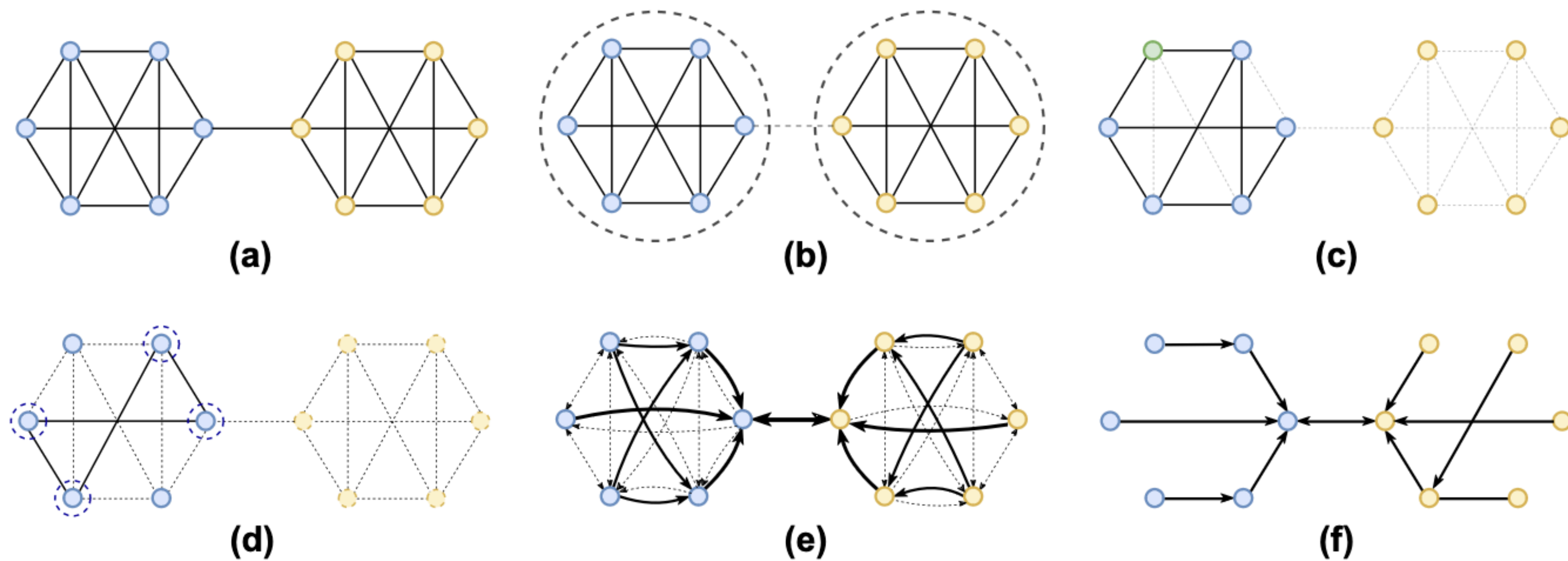


Figure 4: Figure (a) shows a very simple synthetic graph where each node has a binary classification task of determining whether there exists a node of the opposite color in the graph. This task requires learning long-range dependencies. Figure (b) shows a natural clustering of the graph, this clustering would mean no node can do its task if models are trained only on the clusters. Figure (c) shows a neighbor sampling starting from the green node, where random sampling fails to select the important edge that bridges the between the two different colored nodes. Figure (d) shows a random subset sampling strategy that a subset of nodes are likely to be selected from the same color only, making it impossible to reason about the task. (e) shows attention scores between the nodes if trained with an attention based network. Dashed lines have near zero attention scores, and thicker the lines the attention score is higher. Knowing these attention scores will mean each node with just one directional edge can do the task perfectly. The attention edges are shown in (f). In case two nodes are equally informative selecting either of them leads to the correct result.

Table 1: Comparison of our model with other GNNs on five homophilic and three heterogeneous datasets (shown in the left and right tables, respectively). The reported metric is ROC-AUC for the Minesweeper and Tolokers datasets, and accuracy for all others.

Model	Computer	Photo	CS	Physics	ogbn-arxiv	Model	Actor	Minesweeper	Tolokers
GCN	89.65 ± 0.52	92.70 ± 0.20	92.92 ± 0.12	96.18 ± 0.07	71.74 ± 0.29	GLOGNN	36.4 ± 1.6	51.08 ± 1.23	73.39 ± 1.17
GRAPHSAGE	91.20 ± 0.29	94.59 ± 0.14	93.91 ± 0.13	96.49 ± 0.06	71.49 ± 0.27	SGC	27.0 ± 0.9	-	-
GAT	90.78 ± 0.13	93.87 ± 0.11	93.61 ± 0.14	96.17 ± 0.08	72.01 ± 0.20	GCN	33.23 ± 1.16	89.75 ± 0.52	83.64 ± 0.67
GRAPHSAINTE	90.22 ± 0.15	91.72 ± 0.13	94.41 ± 0.09	96.43 ± 0.05	68.50 ± 0.23	GRAPHGPS	-	90.63 ± 0.67	83.71 ± 0.48
NODEFORMER	86.98 ± 0.62	93.46 ± 0.35	95.64 ± 0.22	96.45 ± 0.28	59.90 ± 0.42	NAGPHORMER	-	84.19 ± 0.66	78.32 ± 0.95
GRAPHGPS	91.19 ± 0.54	95.06 ± 0.13	93.93 ± 0.12	97.12 ± 0.19	70.92 ± 0.04	NODEFORMER	36.9 ± 1.0	86.71 ± 0.88	78.10 ± 1.03
GOAT	90.96 ± 0.90	92.96 ± 1.48	94.21 ± 0.38	96.24 ± 0.24	72.41 ± 0.40	GOAT	-	81.09 ± 1.02	83.11 ± 1.04
EXPHORMER+GCN	91.59 ± 0.31	95.27 ± 0.42	95.77 ± 0.15	97.16 ± 0.13	72.44 ± 0.28	EXPHORMER+GAT	38.68 ± 0.38	90.74 ± 0.53	83.77 ± 0.78
EXPHORMER	91.16 ± 0.26	95.36 ± 0.17	95.19 ± 0.26	96.40 ± 0.20	71.27 ± 0.27	EXPHORMER	39.01 ± 0.69	92.26 ± 0.56	83.53 ± 0.28
SPEXPHORMER	90.93 ± 0.08	95.24 ± 0.12	95.00 ± 0.15	96.70 ± 0.05	70.82 ± 0.24	SPEXPHORMER	38.44 ± 0.42	90.72 ± 0.06	83.15 ± 0.12

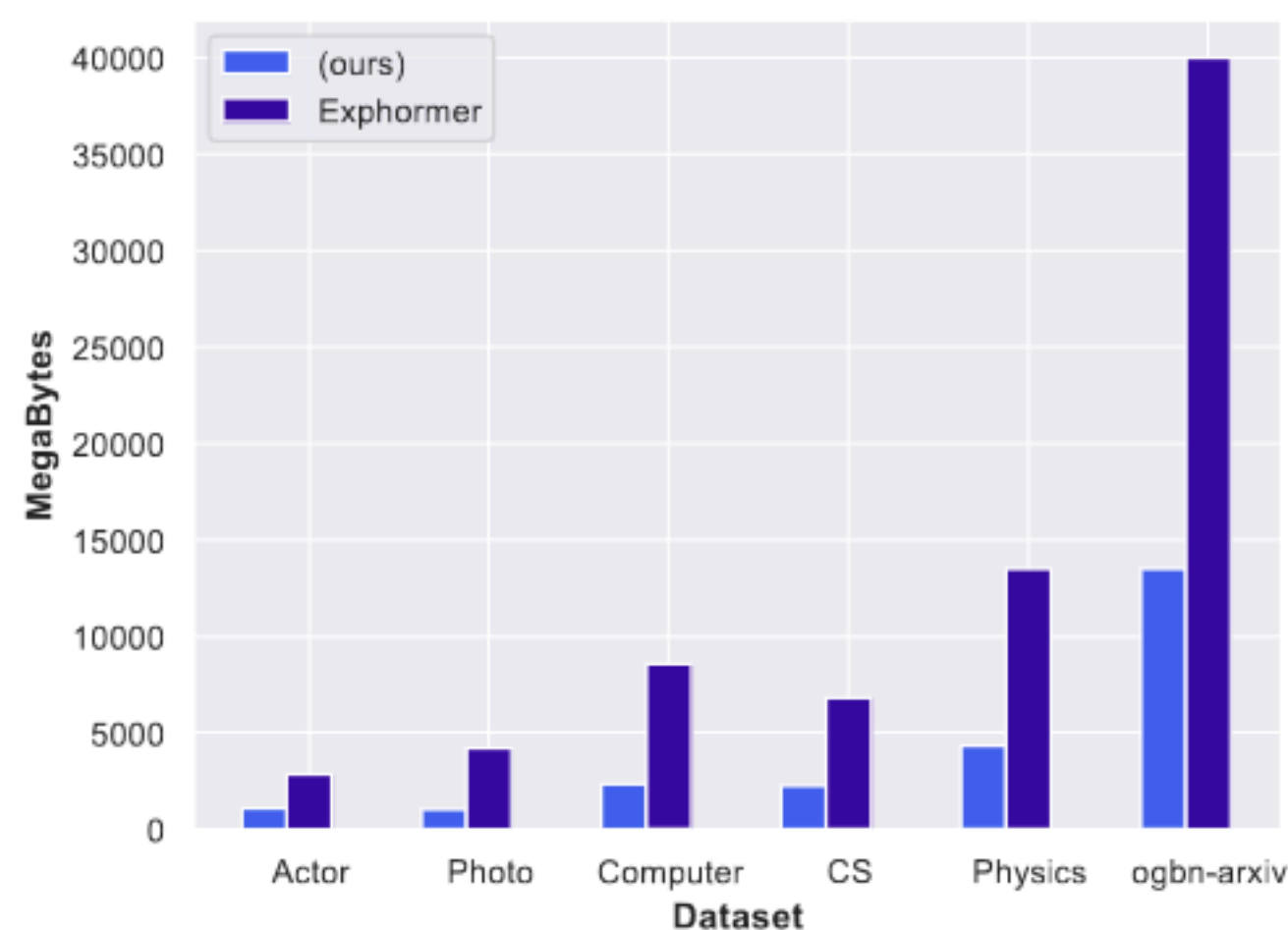


Figure 3: Comparing memory usage of our model with the Exphormer with expander degree 30.

Model	ogbn-proteins	Amazon2M	Pokec
MLP	72.04 \pm 0.48	63.46 \pm 0.10	60.15 \pm 0.03
GCN	72.51 \pm 0.35	83.90 \pm 0.10	62.31 \pm 1.13
SGC	70.31 \pm 0.23	81.21 \pm 0.12	52.03 \pm 0.84
GCN-NSAMPLER	73.51 \pm 1.31	83.84 \pm 0.42	63.75 \pm 0.77
GAT-NSAMPLER	74.63 \pm 1.24	85.17 \pm 0.32	62.32 \pm 0.65
SIGN	71.24 \pm 0.46	80.98 \pm 0.31	68.01 \pm 0.25
NODEFORMER	77.45 \pm 1.15	87.85 \pm 0.24	70.32 \pm 0.45
SGFORMER	79.53 \pm 0.38	89.09 \pm 0.10	73.76 \pm 0.24
SPEXPFORMER	80.66 \pm 0.21	90.32 \pm 0.01	74.73 \pm 0.04

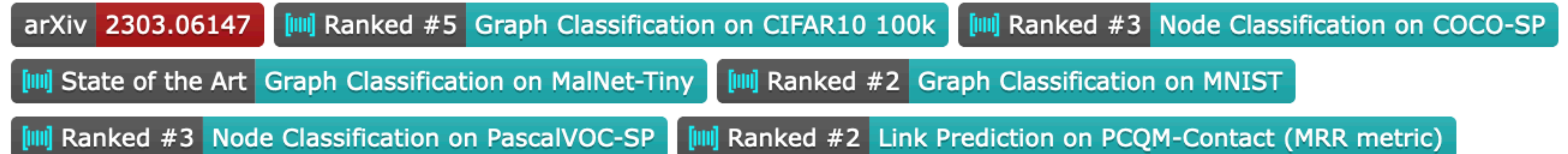
Table 2: Comparative results on large graph datasets, with ROC-AUC reported for the ogbn-proteins dataset and accuracy for all others.

Thanks!

- Graph transformers: like regular transformers, but on graphs
- Full pairwise attention way too expensive for large graphs

Exphormers: Sparse Transformers for Graphs

- Exphormer



- Augment attention graph with expanders + maybe virtual nodes
- Spexphormer
 - Sparsify the augmented graph with a pilot network