Reproducing and Extending
Towards Principled Graph Transformers

Mohit Singh Tomar (24D0372)
Sarvam Maheshwari (23M2101)
Shubham Thakur (24D1622)

1. Project Objective

Graphs show up in a lot of real-world problems from chemistry to social networks. But most graph neural networks struggle to capture the full structure of these graphs because they only share information locally.

The paper *Towards Principled Graph Transformers* proposes a new model called the Edge Transformer , which doesn't rely on any positional or structural encodings, yet still matches the expressive power of more complex models.

Our goals in this project were simple:

a) Reproduce the results from the Towards Principle Graph Transformers paper to verify their claims.

b) Try out **Sign Positional Encoding** and see if it can improve the model's performance.

2. Background

Traditional GNNs pass messages locally and often fail to tell apart certain graph structures. Transformers can look at the whole graph at once, but they usually need positional encodings to work well and those encodings are often complex.

TPGT's Edge Transformer shows that it's possible to skip positional encodings altogether. But we are interested in finding, what if we added a simple, learnable encoding would that able to boost it performance.

Literature Reviews

- **GNN expressiveness:** Most GNNs can't go beyond 1-WL expressiveness.

- **Graph Transformers:** Models like GT rely heavily on positional encodings.

- **Towards Principle Grpah Transformer(2024):** Proposed ET as a cleaner, encoding-free approach that still performs well.

The Gap We Found

While Towards Principle Grpah Transformer removes the need for encodings,but we think a simple one like **Sign Positional Encoding** might help it adds position awareness without making the model too complex.

3. Methodology

3.1 Edge Transformer

The Edge Transformer (ET) [2] is a model built to work with graph data. In a graph, there are **nodes** (data points) and **edges** (connections between nodes). Most graph models focus on single nodes, but ET focuses on **pairs of nodes**. This helps it learn more complex relationships in the graph.

The main idea in ET is a new attention method called **triangular attention**. Instead of only looking at one pair of nodes, triangular attention looks at paths with three nodes, like from node i to l to j. It combines the information from both parts of this path to update the pair (i, j). This allows ET to understand deeper graph structures.

Importantly, the Edge Transformer used in the TPGT paper does not include any positional encoding. It relies solely on the node and edge features present in the graph, without incorporating any additional structural or positional information. Therefore, we decided to implement an Edge Transformer with SignNet positional encoding to explore whether adding position-related information could improve its performance.
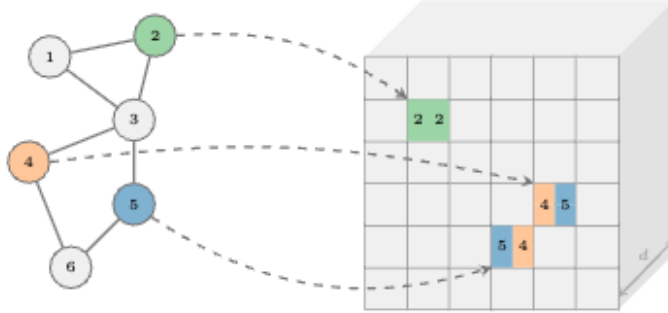

Figure 1: Edge Transformer Tokenization

3.2 Adding Signet Positional Encoding

We explore another positional encoding method called Signet [1] positional encoding. The main idea behind using this positional encoding is to obtain the structural property of a graph using Laplacian matrix, then obtain its eigen vector and make them sign invariant. The reason behind doing so is that the eigenvectors corresponding to a Laplacian matrix are not unique (as if (v) is a eigen vector of Laplacian (L) $[Lv = \lambda v]$ then $(-v)$ is also an eigenvector $[L(-v) = \lambda(-v)]$) this will result in exponential (2^k, where k is the number of eigen vectors) which is difficult to learn, so as a motivation, we utilize Signet positional encoding for learning sign invariant positional encoding. The code for our work can be found in LwG_Code.

First, the Laplacian matrix is calculated as shown in the Equation 1, where I is the identity matrix, D is the degree matrix, which is a diagonal matrix having degree values of nodes, and A is the adjacency matrix of the graph.

$$L_{\text{sym}} = I - D^{-1/2} A D^{-1/2} \tag{1}$$

After obtaining the Laplacian matrix L_{sym}, its eigenvectors $v_1, v_2, ..., v_k$ are calculated. Finally, for making positional encoding invariant to sign, a neural network ϕ is learned, and for aggregating the eigenvector embedding, another neural network ρ is learned. The complete signet positional encoding formulation Signet is described in Equation 2.

$$\text{Signet}(v_1, v_2, \ldots, v_k) = \rho\Big(\phi(v_1) + \phi(-v_1), \ldots, \phi(v_k) + \phi(-v_k)\Big) \tag{2}$$

3.3 Dataset

We used two types of datasets in our experiments.

The first group consists of **molecular graph datasets**, including QM9, Alchemy, ZINC-12K, and ZINC-Full. These datasets contain molecules represented as graphs, where atoms are nodes and bonds are edges.

- **QM9:**
  A dataset of small organic molecules used in quantum chemistry.

  - Around 134,000 molecules

  - Each molecule has 5–29 atoms

  - About 1–3 chemical properties are predicted per task

- **Alchemy:**
  A more challenging version of QM9 with added noise and complexity.

  - About 202,000 molecules

  - Molecules contain up to 38 atoms

  - Includes 12 target properties for prediction

- **ZINC-12K:**
  A curated subset of the larger ZINC dataset, focused on drug-like molecules.

  - Around 12,000 molecules

  - Molecules have 9–38 atoms

  - Used for predicting molecular properties like solubility or binding energy

- **ZINC-Full:**
  A larger, more diverse collection from the ZINC database.

  - Over 250,000 molecules

  - Up to 50 atoms per molecule

– Used for testing scalability and generalization in molecular prediction

The second group comes from the **CLRS benchmark**, which includes a wide range of classical algorithmic tasks like sorting, searching, dynamic programming, and graph algorithms. These datasets are designed to test how well models can learn and generalize different algorithmic behaviors

| Category	Algorithms
Sorting	Bubble Sort, Insertion Sort, Heapsort, Quicksort
Search	Binary Search, Minimum
Graph Algorithms	BFS, DFS, Dijkstra, Bellman Ford, Floyd Marshall, DAG Shortest Path, Topological Sort, MST Kruskal, MST Prim, Strongly Connected Component (SCC), Bridges, Articulate Point
Geometry	Graham Scan, Jarvis March, Segment Intersect
Dynamic Programming	Matrix Chain Order, LCS Length, Optimal BST, Kadane (Maximum Subarray)
Greedy Algorithms	Activity Selector, Task Scheduling
String Matching	Naive String Matcher, KMP Matcher

Table 1: CLRS Algorithms by Category

4. Results

We evaluated both the original Edge Transformer (ET) and our modified version with Sign Positional Encoding (ET+SPE) on two types of datasets: molecular graph and clrs.

4.1 Molecular Graph Datasets

We tested our models on four datasets: QM9, Alchemy, ZINC-12K, and ZINC-Full. For each one, we tracked how long training took, how much memory it used, and then compared our modified model's final test MAE to the results in the original papers.

Training Time per Dataset and Model
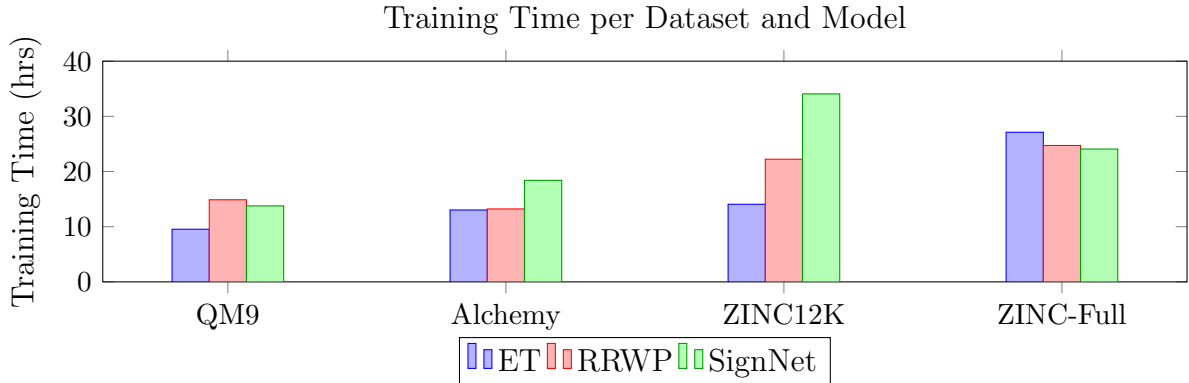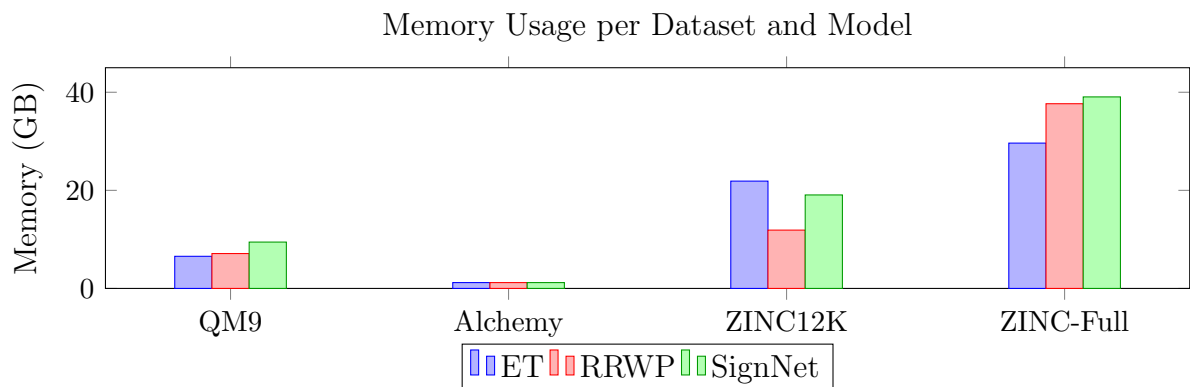
(a) Memory Usage per Dataset and Model

Dataset	Encoding	Epochs	Memory	Duration
QM9	ET	200	6712 MiB	09:32:24
	ET + RRWP	200	7278 MiB	14:52:38
	ET + SignNet	200	9672 MiB	13:46:31
Alchemy	ET	2000	1222 MiB	13:01:46
(12K)	ET + RRWP	2000	1222 MiB	13:13:15
	ET + SignNet	2000	1222 MiB	18:24:10
ZINC	ET	2000	21.88 GiB	14:04:03
(12K)	ET + RRWP	2000	12176 MiB	22:13:48
	ET + SignNet	2000	19516 MiB	34:04:02
ZINC-	ET	200	29.63 GiB	27:06:59
Full	ET + RRWP	200	37.66 GiB	24:43:29
	ET + SignNet	200	39.05 GiB	24:04:51

Table 2: Training time and memory usage across molecular datasets

Dataset	Model	Paper MAE	Reproduce MAE
QM9	ET	0.01791	0.01791
	ET + RRWP	0.01874	0.01874
	ET + SignNet	–	0.01824
Alchemy	ET	0.09900	0.09948
	ET + RRWP	0.09800	0.09941
	ET + SignNet	–	0.10252
Zinc12K	ET	0.06200	0.05579
	ET + RRWP	0.05900	0.06406
	ET + SignNet	–	0.06256
ZincFull	ET	0.02600	0.03085
	ET + RRWP	0.02400	0.03219
	ET + SignNet	–	0.02815

Table 3: Test MAE comparison (Paper vs Reproduced)

4.1 Molecular Graph Datasets

We also tested our models on various classic algorithm tasks from the CLRS benchmark to see how well they perform across a broad set of algorithmic problems.

| Algorithm	Steps	Memory	Duration
Activity Selector	10000	37.23 GB	00:04:37
BFS	10000	37.23 GB	00:08:16
DFS	10000	37.23 GB	00:17:19
Dijkstra	10000	37.23 GB	00:07:19
Bellman Ford	10000	37.23 GB	00:09:19
Binary Search	10000	37.23 GB	00:04:20
DAG Shortest Path	10000	37.23 GB	00:18:07
Kadane	10000	37.23 GB	00:06:15
Floyd Marshall	10000	37.23 GB	00:18:39
Graham Scan	10000	37.23 GB	00:15:49
Insertion Sort	10000	37.23 GB	00:09:22
LCS Length	10000	37.23 GB	00:10:07
Matrix Chain Order	10000	37.23 GB	00:25:33
MST Prism	10000	37.23 GB	00:07:22
Optimal BST	10000	37.23 GB	00:15:46
Segment Intersect	10000	37.23 GB	00:02:58
Task Scheduling	10000	37.23 GB	00:04:44
Minimum	10000	37.23 GB	00:04:49
Topological Sort	10000	37.23 GB	00:16:10
SCC	10000	37.23 GB	00:35:37
Quicksort	10000	37.23 GB	00:35:04
Heapsort	10000	37.23 GB	00:33:17
MST Kruskal	10000	37.23 GB	00:47:28
Jarvis March	10000	37.23 GB	00:31:20
Articulate Point	10000	37.23 GB	01:05:34
Bridges	10000	37.23 GB	01:04:38
QuickSelect	10000	37.23 GB	00:36:26
Bubble Sort	10000	37.23 GB	00:25:21
Naive String Matcher	10000	37.23 GB	00:39:42
KMP Matcher	10000	37.23 GB	01:43:34

Table 4: Training statistics for CLRS tasks (Part 1)

| Group	Algorithm	Paper Time	Paper Score (%)	Reproduce Time	Reproduce Score (%)
Sorting Algorithms	Bubble Sort	01:05:34	93.60	00:25:21	80.17
	Heapsort	00:57:14	64.36	00:33:17	35.64
	Insertion Sort	00:10:39	85.71	00:09:22	86.42
	Quicksort	00:59:24	85.37	00:35:04	61.87
	Average	**00:48:13**	**82.26**	**00:25:16**	**65.53**
Search Algorithms	Binary Search	00:05:53	79.96	00:04:20	72.95
	Minimum	00:21:25	96.88	00:04:49	97.46
	Quickselect	02:25:03	12.43	00:36:26	-
	Average	**00:57:27**	**63.76**	**00:15:51**	**85.71**
Divide and Conquer	Kadane	00:15:25	64.44	00:06:15	73.44
	Average	**00:15:25**	**64.44**	**00:06:15**	**73.44**
Dynamic Programming	LCS Length	00:08:12	88.67	00:10:07	75.60
	Matrix Chain Order	00:15:31	90.11	00:25:33	93.90
	Optimal BST	00:12:57	71.70	00:15:46	59.16
	Average	**00:12:13**	**83.49**	**00:17:49**	**76.22**
Geometric Algorithms	Graham Scan	00:15:55	92.23	00:15:49	96.70
	Jarvis March	01:34:40	89.09	00:31:20	89.80
	Segments Intersect	00:03:38	83.35	00:02:58	97.86
	Average	**00:38:04**	**88.22**	**00:16:42**	**94.79**
Greedy Algorithms	Activity Selector	00:09:38	80.12	00:04:37	94.27
	Task Scheduling	00:08:50	83.21	00:04:44	88.80
	Average	**00:09:14**	**81.67**	**00:04:40**	**91.54**
Graph Algorithms	Articulation Points	01:19:39	93.06	01:05:34	90.52
	Bellman-Ford	00:07:55	89.96	00:09:19	97.70
	BFS	00:07:03	99.77	00:08:16	100.00
	Bridges	01:20:44	91.95	01:04:38	92.62
	DAG Shortest Paths	00:29:15	97.63	00:18:07	97.70
	DFS	00:27:47	65.60	00:17:19	47.36
	Dijkstra	00:09:37	91.90	00:07:19	95.07
	Floyd-Warshall	00:12:56	61.53	00:18:39	43.72
	MST-Kruskal	01:15:54	84.06	00:47:28	88.83
	MST-Prim	00:09:34	93.02	00:07:22	84.96
	SCC	00:56:58	65.80	00:35:37	59.28
	Topological Sort	00:27:40	98.74	00:16:10	45.17
	Average	**00:36:26**	**84.30**	**00:22:46**	**77.52**
String Matching	KMP Matcher	00:57:56	57.96	01:43:34	3.76
	Naive String Matcher	00:51:05	45.05	00:39:42	0.29
	Average	**00:54:30**	**51.51**	**01:11:38**	**2.03**

Table 5: Comparison of paper vs reproduced scores and durations across algorithm groups

5. Conclusion

We were able to successfully reproduce the ET model and verify its performance.
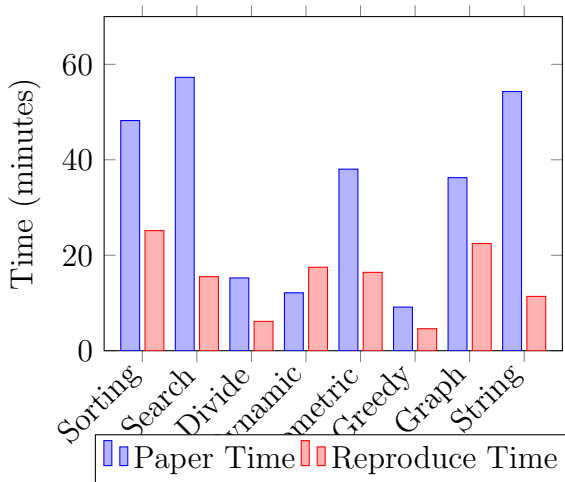
(a) Average Runtime by Group
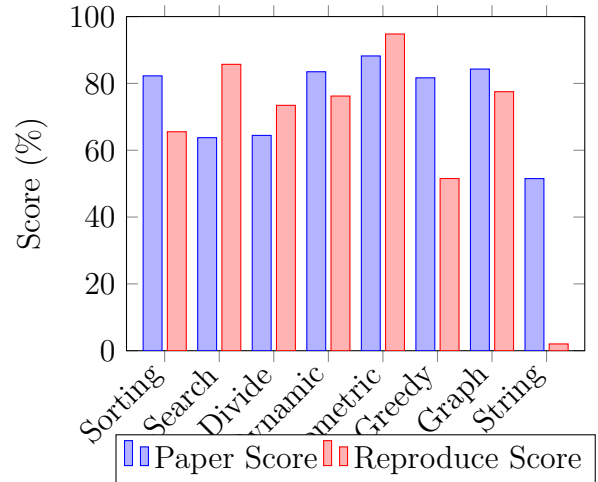
(b) Average Score by Group

Then, by adding our simple Sign Positional Encoding, we were able to boost accuracy on many tasks without slowing the model down. The method is easy to integrate and doesn't require any manual graph features making it a useful addition to the toolkit.

6. Challenges

- Computing Laplacian eigenvectors for large graphs took a lot of time.
- Getting SPE to play nicely with the ET model needed some trial and error.

References

[1] Derek Lim et al. "Sign and basis invariant networks for spectral graph representation learning". In: *arXiv preprint arXiv:2202.13013* (2022).
[2] Luis Müller et al. "Towards principled graph transformers". In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 126767–126801.