Leveraging Classical Algorithms for Graph Neural Networks

Jason Wu*
University of Cambridge
jw2313@cantab.ac.uk

Petar Veličković
Google DeepMind / University of Cambridge
petarv@google.com

Abstract

Neural networks excel at processing unstructured data but often fail to generalise out-of-distribution, whereas classical algorithms guarantee correctness but lack flexibility. We explore whether pretraining Graph Neural Networks (GNNs) on classical algorithms can improve their performance on molecular property prediction tasks from the Open Graph Benchmark: *ogbg-molhiv* (HIV inhibition) and *ogbg-molclintox* (clinical toxicity). GNNs trained on 24 classical algorithms from the CLRS Algorithmic Reasoning Benchmark are used to initialise and freeze selected layers of a second GNN for molecular prediction. Compared to a randomly initialised baseline, the pretrained models achieve consistent wins or ties, with the *Segments Intersect* algorithm pretraining yielding a 6% absolute gain on *ogbg-molhiv* and *Dijkstra* pretraining achieving a 3% gain on *ogbg-molclintox*. These results demonstrate embedding classical algorithmic priors into GNNs provides useful inductive biases, boosting performance on complex, real-world graph data.

1 Introduction

Neural networks and classical algorithms are often seen as distinct paradigms for problem-solving. Neural networks are highly effective at processing unstructured data, enabling a single model to handle diverse tasks without explicit programming [1]. However, they often struggle to generalise beyond the distribution of their training data, leading to unreliable behaviour on larger or unseen inputs [2]. Classical algorithms, on the other hand, follow precise logical rules that guarantee correctness regardless of input size [3], but their dependence on well-defined input structures makes them less adaptable to new tasks.

This dichotomy raises an interesting question: can we leverage classical algorithms to improve the performance of a Graph Neural Network (GNN) on real-world tasks?

We address this by studying two molecular property prediction benchmarks from the Open Graph Benchmark (OGB) [4]: *ogbg-molhiv*, which predicts whether a molecule inhibits HIV replication, and *ogbg-molclintox*, which predicts a molecule's clinical toxicity. In both cases, molecules are represented as graphs, with atoms as nodes and chemical bonds as edges.

Our hypothesis is that learning to execute classical algorithms can endow a GNN with inductive biases that improve property prediction. For example, by learning to identify shortest paths, the model can learn to identify nodes that frequently lie along shortest paths. Such nodes can correspond to critical chemical substructures, such as functional groups, that strongly influence molecular behaviour.

To test this, we first train GNNs—the *CLRS models*—on a variety of algorithms from the CLRS Algorithmic Reasoning Benchmark [5]. Their learnt weights are then transferred to selected layers of a second GNN (*OGB model*) trained on *ogbg-molhiv* and *ogbg-molclintox*. Compared to a randomly initialised baseline with the same architecture, pretraining yields up to a 6% absolute improvement—equivalent to discovering roughly 60 additional active compounds per $1{,}000$ screened, potentially saving substantial experimental effort.

*Now at Google.

Wu and Veličković, Leveraging Classical Algorithms for Graph Neural Networks (Extended Abstract). Presented at the Fourth Learning on Graphs Conference (LoG 2025), Hybrid Event, December 10–12, 2025.

Recent work on *graph foundation models* [6] pretrains GNNs on large-scale datasets before adapting them to downstream tasks [7]. Our approach is complementary: rather than relying primarily on scale, we pretrain on well-specified algorithmic tasks to inject inductive biases that transfer to molecular property prediction. It is also more efficient—training a model on an algorithm for 10,000 steps takes just 2 hours on a single L4 GPU, compared to 6.4 days on eight A40 GPUs for GraphFM [8]. Similarly, while GraphMAE requires 24 GB GPUs [9], our method fits comfortably within 8 GB of memory. Even the generalist variant that trains on all 24 algorithms concurrently completes within 1.4 days on an A100 GPU.

Our result sits nicely within the realm of prior deployments of CLRS training within more specialised architectures—from reinforcement learning agents [10, 11] to self-supervised feature extractors [12] and primal-dual solvers [13]. To the best of our knowledge, we present the first result of this kind over a *generic* GNN pipeline deployed over a standard supervised learning dataset such as *ogbg-molhiv*.

2 CLRS Model

Our CLRS model follows an *encode–process–decode* framework. In this framework, the input data is first transformed into a latent representation, then processed through iterative computation, and finally decoded to generate predictions.

2.1 Encode

Let $\mathbf{x}_i$ denote the raw features of node i, $\mathbf{e}_{ij}$ the raw features of the edge from node i to node j, and $\mathbf{g}$ the raw features representing the entire graph. In the encoding stage, these raw features are transformed into high-dimensional embeddings using learnable encoder functions specific to each algorithm:

$$\mathbf{h}_i = f_n(\mathbf{x}_i), \quad \mathbf{h}_{ij} = f_e(\mathbf{e}_{ij}), \quad \mathbf{h}_g = f_g(\mathbf{g}).$$

Here, f_n, f_e, and f_g are typically implemented as linear layers that map the raw inputs into a unified latent space. This unified representation allows the model to work with heterogeneous inputs in a consistent manner.

2.2 Process

During the processing stage, the encoded features are refined by a processor network that simulates the operations of a classical algorithm. A common approach is to split this process up to *Message Computation*, *Message Aggregation* and *Feature Update*.

Message Computation: Messages are computed along each edge using a message function f_m. For each edge (i, j), a message is computed as:

$$\mathbf{m}_{ij} = f_m\big(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_{ij}, \mathbf{h}_g\big).$$

Message Aggregation: These messages are then aggregated for each node i using a permutation-invariant operator, denoted as $\bigoplus$:

$$\mathbf{m}_i = \bigoplus_{j \in \mathcal{N}_i} \mathbf{m}_{ji},$$

where $\mathcal{N}_i$ represents the set of neighbours of node i. Permutation invariance ensures that the order of messages does not affect the result.

Feature Update: Finally, the aggregated message $\mathbf{m}_i$ is combined with the original node embedding $\mathbf{h}_i$ using a readout function f_r to produce the updated node representation:

$$\mathbf{h}'_i = f_r\big(\mathbf{h}_i, \mathbf{m}_i\big).$$

This iterative processing step can be repeated multiple times to allow the network to capture both local and global dependencies.

2.3 Decode

Once the node embeddings have been updated, they are converted into task-specific predictions through a decoding stage. A decoder function $g.$ maps the processed embeddings to the desired output space:

$$\hat{\mathbf{y}}_i = g(\mathbf{h}'_i).$$

3 OGB Model

Our OGB model adopts a similar *encode–process–decode* framework as the CLRS model, illustrated in Figure 1. We fix our processor as the Triplet-GMPNN, since Ibarz et al. [14] have demonstrated that this GNN yields strong average test performance on executing CLRS algorithms.

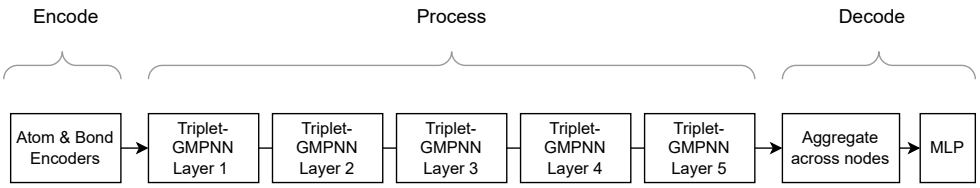


Figure 1: OGB Model Architecture.

The *process* stage comprises five stacked Triplet-GMPNN layers, where each layer's output feeds into the next. To assess the effect of algorithmic priors, we compare the *pretrained* model with the *baseline* of identical architecture. In the baseline, all layers are randomly initialised and trainable; in the pretrained model, layers 2 and 4 are initialised with CLRS weights and kept frozen.

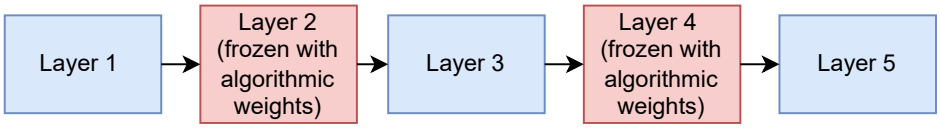


Figure 2: *Pretrained* OGB model, with blue denoting trainable layers and red denoting frozen layers initialised from algorithmic weights.

We chose this alternating freezing strategy to balance retaining algorithmic priors with flexibility for adaptation, as done previously in graph representation learning approaches such as [10]. Alternative strategies—freezing early layers (1–2) or fully fine-tuning all pretrained layers—yielded less pronounced results for *ogbg-molhiv*, achieving top accuracies of 75.05% and 76.52%, respectively, compared to 77.16% with the alternating strategy (see appendix for full results). We hypothesise that freezing early layers may limit how well the pretrained knowledge integrates, while fine-tuning all layers could overwrite it. We therefore focus on the alternating strategy in our main evaluation.

4 Evaluation

In this section, we evaluate the impact of pretraining on different algorithms. We focus on the 24 algorithms for which the Triplet-GMPNN achieves test accuracies exceeding 50% [14]—presuming that, in those cases, the model has learnt meaningful algorithmic representations. The corresponding validation results in Table 2 (Appendix) confirm that most algorithms were learned effectively.

To compare the performance of the pretrained models against the baseline model, we employ a win/tie/loss metric. This metric is defined as follows:

Let $\mu(M)$ and $\sigma(M)$ denote the mean and standard deviation of model M's test performance, respectively. Model X is said to win over model Y if

$$\mu(X) - \sigma(X) > \mu(Y).$$

If this inequality does not hold, the result is considered a tie (T).

For the *ogbg-molhiv* task, the *Segments Intersect* algorithm yields the largest gain—an absolute improvement of approximately 6% over the baseline—achieving a test accuracy of 77.16%[2]. A Welch's *t*-test across five runs confirms the improvement is statistically significant: the pretrained model achieved $76.63 \pm 1.01\%$ versus the baseline's $70.94 \pm 0.80\%$ (t = 9.40, p = 2.9×10^{-4}). Results from the early-layer freezing strategy in Table 3 further support our takeaways.

[2]For context: a state-of-the-art GNN architecture, GSAT [15], reports $80.67 \pm 0.09\%$ on this dataset.

Table 1: Test performance on *ogbg-molhiv* and *ogbg-molclintox*, reported as mean accuracy $\pm$ standard deviation over three runs. "W" denotes $\mu_{\text{model}} - \sigma_{\text{model}} > \mu_{\text{baseline}}$, "L" denotes $\mu_{\text{baseline}} - \sigma_{\text{baseline}} > \mu_{\text{model}}$, and "T" a tie otherwise.

Algorithm	**MolHIV Acc. (%)**	**MolClinTox Acc. (%)**
Baseline (No algorithms)	71.20 ± 0.68	86.85 ± 2.53
Articulation Points	73.00 ± 1.09 (W)	86.27 ± 0.88 (T)
Activity Selector	72.26 ± 1.34 (T)	84.46 ± 3.56 (T)
Bellman–Ford	73.80 ± 2.85 (T)	87.58 ± 3.98 (T)
BFS	73.47 ± 0.78 (W)	84.54 ± 1.01 (T)
Binary Search	74.09 ± 1.59 (W)	87.21 ± 0.65 (T)
Bridges	71.18 ± 4.53 (T)	86.99 ± 2.64 (T)
Bubble Sort	73.14 ± 0.80 (W)	88.23 ± 1.78 (T)
DAG Shortest Paths	73.14 ± 1.50 (W)	84.05 ± 1.75 (L)
Dijkstra	72.71 ± 2.51 (T)	$\mathbf{90.06 \pm 2.76}$ (W)
Find Maximum Subarray (Kadane)	73.58 ± 1.18 (W)	88.11 ± 1.20 (W)
Graham Scan	73.28 ± 0.95 (W)	88.49 ± 1.69 (T)
Insertion Sort	75.61 ± 3.38 (W)	84.12 ± 2.08 (L)
Jarvis' March	73.01 ± 2.62 (T)	88.41 ± 1.21 (W)
LCS Length	73.59 ± 3.87 (T)	88.02 ± 1.33 (T)
Matrix Chain Order	72.22 ± 2.32 (T)	89.20 ± 0.44 (W)
Minimum	70.87 ± 2.06 (T)	86.51 ± 3.41 (T)
MST Kruskal	76.92 ± 1.28 (W)	90.26 ± 4.38 (T)
MST Prim	74.12 ± 1.59 (W)	85.97 ± 1.88 (T)
Naive String Matcher	75.03 ± 3.05 (W)	86.76 ± 1.75 (T)
Optimal BST	74.86 ± 0.65 (W)	87.35 ± 2.40 (T)
Quicksort	72.74 ± 2.75 (T)	89.68 ± 0.73 (W)
Segments Intersect	$\mathbf{77.16 \pm 0.80}$ (W)	86.57 ± 1.25 (T)
Task Scheduling	72.97 ± 2.59 (T)	89.00 ± 4.69 (T)
Topological Sort	72.36 ± 1.32 (T)	85.03 ± 2.17 (T)
All Algorithms Concurrently	74.70 ± 1.84 (W)	86.54 ± 2.89 (T)

Segments Intersect is a classical computational geometry algorithm [3] that determines whether two line segments intersect by evaluating the orientations of point triplets. In molecular graphs, the three-dimensional arrangement of atoms and bonds is crucial for determining HIV inhibition, as interactions such as hydrophobic effects and steric clashes depend heavily on spatial structure. Pretraining on *Segments Intersect* may therefore provide inductive biases for spatial reasoning, enhancing recognition of relevant structural motifs. The corresponding t-SNE embeddings show building on top of this learned algorithm improves the clustering of active compounds (see Appendix A, Figure 3).

For the *ogbg-molclintox* task, the largest gain comes from *Dijkstra*, which outperforms the baseline by over 3%. This improvement is likely because the shortest-path computation encourages the model to identify atoms that frequently occur along key molecular pathways—features that may correlate with clinical toxicity. In contrast, *DAG Shortest Paths* and *Insertion Sort* perform below the baseline. The former assumes acyclic graph structure, which does not hold for many molecular graphs, and could lead to misaligned inductive biases. The latter focuses on ordering comparisons rather than more generic relational reasoning, providing less relevance to molecular connectivity and potentially introducing noisy representations. Consistent with this, the t-SNE visualisations show that embeddings pretrained on *Dijkstra* exhibit clearer clustering than those from the baseline, while *DAG Shortest Paths* and *Insertion Sort* yield more diffuse distributions (see Appendix A, Figure 4).

Training on all algorithms concurrently still yields a gain over the baseline on *ogbg-molhiv*, though to a lesser extent than *Segments Intersect* alone—likely due to interference from less relevant tasks. Overall, the results support our hypothesis that pretrained algorithmic weights can improve real-world performance, pointing to promising avenues for further research, such as extending to additional graph domains and refining multi-algorithm pretraining strategies.

Acknowledgements

We thank Federico Barbero, Simon Osindero, and Ndidi Elue from Google DeepMind for their helpful feedback and internal reviews during the development of this work.

References

[1] Amir R. Zamir, Alex Sax, William Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722, 2018. 1

[2] Petar Veličković and Charles Blundell. Neural Algorithmic Reasoning. *Patterns*, 2021. 1

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 2009. 1, 4

[4] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hong Ren, Zhitao Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020. 1

[5] Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The CLRS Algorithmic Reasoning Benchmark, 2022. 1

[6] Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Mikhail Galkin, and Jiliang Tang. Position: Graph foundation models are already here. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=Edz0QXKKAo. 2

[7] Jiawei Liu, Cheng Yang, Zhiyuan Lu, Junze Chen, Yibo Li, Mengmei Zhang, Ting Bai, Yuan Fang, Lichao Sun, Philip S. Yu, and Chuan Shi. Graph foundation models: Concepts, opportunities and challenges. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(6):5023–5044, June 2025. ISSN 1939-3539. doi: 10.1109/tpami.2025.3548729. URL http://dx.doi.org/10.1109/TPAMI.2025.3548729. 2

[8] Divyansha Lachi, Mehdi Azabou, Vinam Arora, and Eva Dyer. Graphfm: A scalable framework for multi-graph pretraining, 2024. URL https://arxiv.org/abs/2407.11907. 2

[9] Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. Graphmae: Self-supervised masked graph autoencoders, 2022. URL https://arxiv.org/abs/2205.10803. 2

[10] Andreea-Ioana Deac, Petar Veličković, Ognjen Milinkovic, Pierre-Luc Bacon, Jian Tang, and Mladen Nikolic. Neural algorithmic reasoners are implicit planners. *Advances in Neural Information Processing Systems*, 34:15529–15542, 2021. 2, 3

[11] Yu He, Petar Veličković, Pietro Liò, and Andreea Deac. Continuous neural algorithmic planners. In *Learning on Graphs Conference*, pages 54–1. PMLR, 2022. 2

[12] Petar Veličković, Matko Bošnjak, Thomas Kipf, Alexander Lerchner, Raia Hadsell, Razvan Pascanu, and Charles Blundell. Reasoning-modulated representations. In *Learning on Graphs Conference*, pages 50–1. PMLR, 2022. 2

[13] Danilo Numeroso, Davide Bacciu, and Petar Veličković. Dual algorithmic reasoning. *arXiv preprint arXiv:2302.04496*, 2023. 2

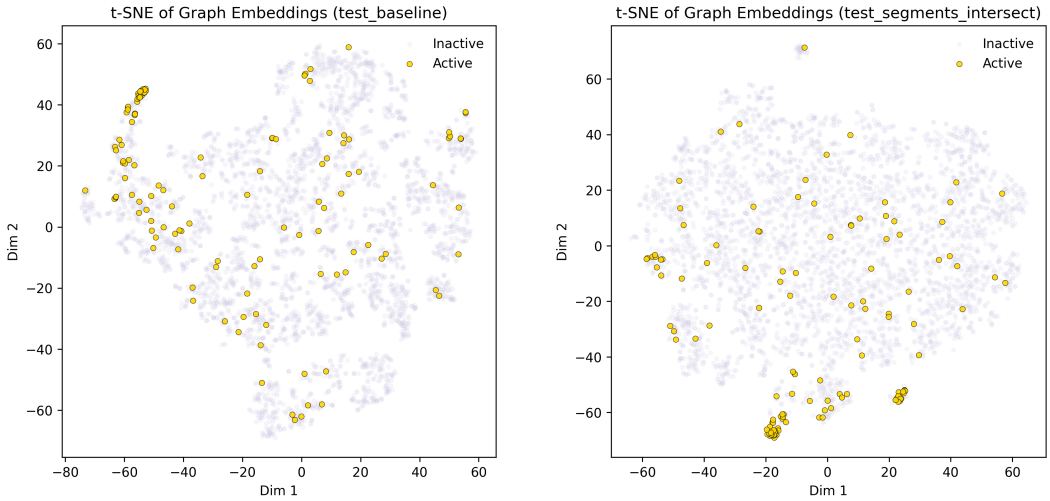
[14] Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, Andreea Deac, Beatrice Bevilacqua, Yaroslav Ganin, Charles Blundell, and Petar Veličković. A generalist neural algorithmic learner, 2022. URL https://arxiv.org/abs/2209.11142. 3, 8

[15] Siqi Miao, Mia Liu, and Pan Li. Interpretable and generalizable graph learning via stochastic attention mechanism. In *International conference on machine learning*, pages 15524–15543. PMLR, 2022. 3

A t-SNE Visualisations of Graph Embeddings

Additional t-SNE visualisations of final graph embeddings are provided here for reference. Figure 3 shows the *ogbg-molhiv* embeddings, and Figure 4 shows the *ogbg-molclintox* embeddings.

In Figure 3, the model pretrained on *Segments Intersect* exhibits more distinct clustering of active compounds, particularly in the lower region, compared to the baseline. This supports our analysis that pretraining on *Segments Intersect* does provide useful inductive biases.

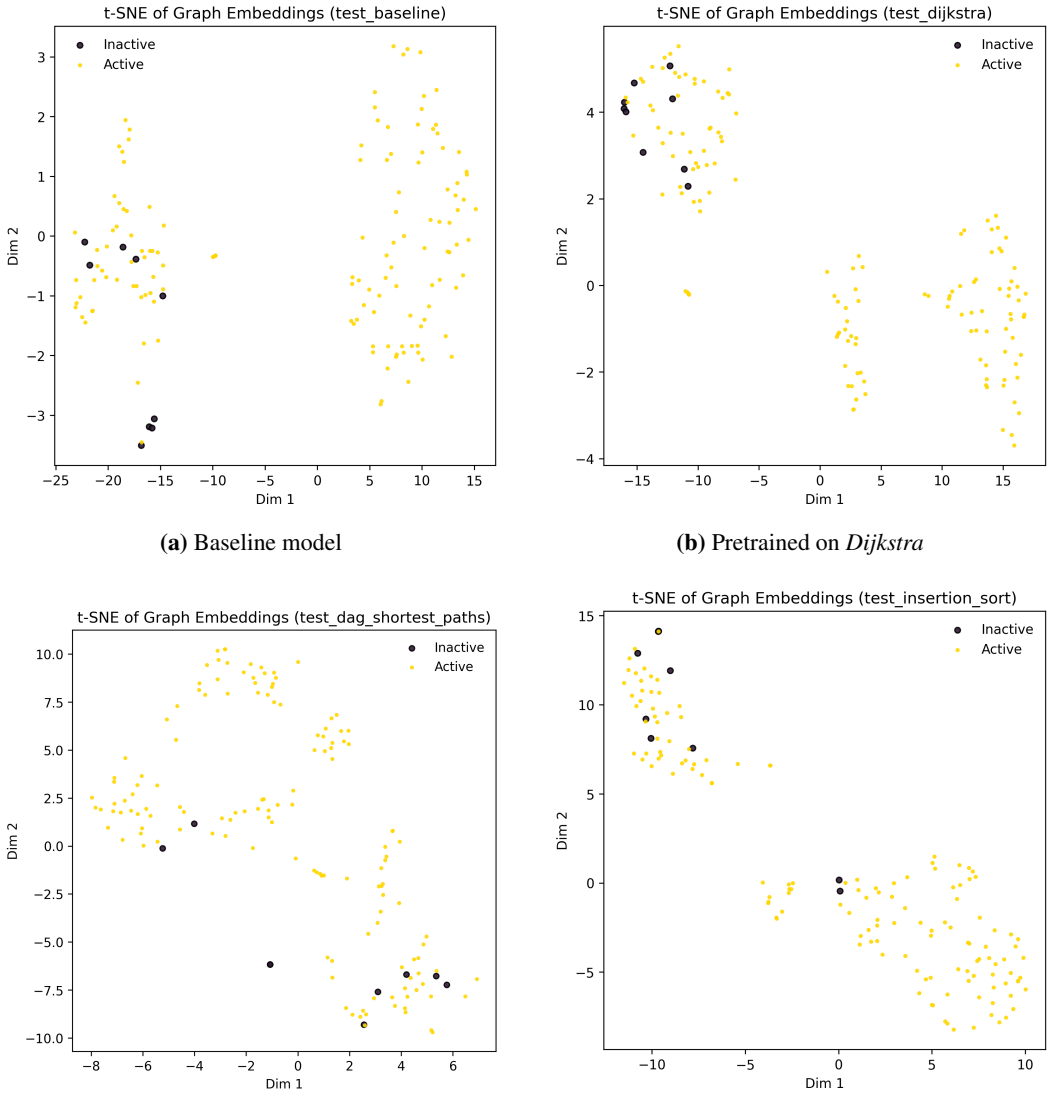


(a) Baseline model

(b) Pretrained on *Segments Intersect*

Figure 3: t-SNE visualisation of graph embeddings for the baseline and pretrained models on *ogbg-molhiv*.

In Figure 4, the model pretrained on *Dijkstra* exhibits more distinct clustering of negative compounds in the top-left region, compared to the baseline and other algorithms like *DAG Shortest Paths* and *Insertion Sort*. This supports our analysis that pretraining on *Dijkstra* does provide useful inductive biases.



(a) Baseline model

(b) Pretrained on *Dijkstra*

(c) Pretrained on *DAG Shortest Paths*

(d) Pretrained on *Insertion Sort*

Figure 4: t-SNE visualisation of graph embeddings for the baseline and pretrained models on *ogbg-molclintox*.

B CLRS Pretraining Validation Accuracy

Here we present the in-distribution validation data of CLRS pretraining, as out-of-distribution test graphs caused GPU memory overflows for some of the algorithms. Most algorithms achieve high validation accuracy, indicating that the *CLRS* models have learnt them effectively. The exception is *Naive String Matcher*, which performs poorly without teacher forcing. This behaviour is consistent with previous observations [14], where teacher forcing improves sorting and Kruskal tasks but degrades performance on string matching.

Table 2: Validation accuracies of the Triplet–GMPNN on CLRS algorithms used for pretraining, reported as mean accuracy $\pm$ standard deviation over three runs.

Algorithm	**Mean Acc. (%)**
Activity Selector	90.85 ± 1.06
Articulation Points	99.79 ± 0.21
Bellman–Ford	99.84 ± 0.05
BFS	100.00 ± 0.00
Binary Search	96.55 ± 0.25
Bridges	100.00 ± 0.00
Bubble Sort	98.70 ± 0.26
DAG Shortest Paths	99.97 ± 0.05
Dijkstra	99.93 ± 0.05
Find Maximum Subarray (Kadane)	97.24 ± 0.17
Graham Scan	97.12 ± 0.58
Insertion Sort	99.02 ± 0.60
Jarvis' March	92.09 ± 0.71
LCS Length	95.46 ± 0.48
Matrix Chain Order	99.50 ± 0.06
Minimum	99.60 ± 0.14
MST Kruskal	99.21 ± 1.11
MST Prim	99.61 ± 0.16
Naive String Matcher	38.57 ± 0.77
Optimal BST	96.59 ± 0.12
Quicksort	98.86 ± 0.39
Segments Intersect	89.41 ± 7.49
Task Scheduling	98.33 ± 0.28
Topological Sort	99.61 ± 0.14

C Freeze Early Layers Performance

Table 3 reports the test performance with early-layer freezing. *Segments Intersect* achieves the best results on both *ogbg-molhiv* and *ogbg-molclintox*, reinforcing our earlier finding that it provides useful inductive biases for molecular property prediction.

Table 3: Test performance on *ogbg-molhiv* and *ogbg-molclintox* with early-layer freezing, reported as mean accuracy $\pm$ standard deviation over three runs. "W" denotes $\mu_{\text{model}} - \sigma_{\text{model}} > \mu_{\text{baseline}}$, "L" denotes $\mu_{\text{baseline}} - \sigma_{\text{baseline}} > \mu_{\text{model}}$, and "T" a tie otherwise.

Algorithm	**MolHIV Acc. (%)**	**MolClinTox Acc. (%)**
Baseline (No algorithms)	71.20 ± 0.68	86.85 ± 2.53
Activity Selector	70.76 ± 2.46 (T)	85.91 ± 0.74 (T)
Articulation Points	71.52 ± 2.02 (T)	82.25 ± 1.84 (L)
Bellman–Ford	74.77 ± 0.75 (W)	85.25 ± 3.45 (T)
BFS	72.39 ± 1.46 (T)	82.47 ± 0.21 (L)
Binary Search	70.65 ± 1.39 (T)	87.05 ± 4.00 (T)
Bridges	69.95 ± 2.40 (L)	83.54 ± 1.50 (L)
Bubble Sort	71.70 ± 0.30 (W)	86.32 ± 2.18 (T)
DAG Shortest Paths	70.77 ± 1.68 (T)	87.70 ± 0.41 (W)
Dijkstra	73.38 ± 0.95 (W)	84.14 ± 4.34 (L)
Find Maximum Subarray (Kadane)	70.84 ± 1.62 (T)	87.48 ± 2.57 (T)
Graham Scan	72.44 ± 0.39 (W)	83.17 ± 3.25 (L)
Insertion Sort	74.26 ± 0.88 (W)	88.68 ± 3.01 (T)
Jarvis' March	70.40 ± 2.27 (L)	85.00 ± 0.46 (T)
LCS Length	72.00 ± 1.41 (T)	87.08 ± 1.69 (T)
Matrix Chain Order	72.47 ± 0.45 (W)	84.44 ± 2.85 (T)
Minimum	71.36 ± 3.08 (T)	85.98 ± 3.56 (T)
MST Kruskal	71.99 ± 0.48 (W)	87.90 ± 3.41 (T)
MST Prim	73.90 ± 1.07 (W)	85.23 ± 1.89 (T)
Naive String Matcher	72.22 ± 2.70 (T)	85.23 ± 2.15 (T)
Optimal BST	72.03 ± 1.61 (T)	84.30 ± 3.23 (L)
Quicksort	71.85 ± 0.66 (T)	85.21 ± 2.07 (T)
Segments Intersect	$\mathbf{75.05 \pm 1.45}$ (W)	$\mathbf{89.15 \pm 1.37}$ (W)
Task Scheduling	73.46 ± 0.82 (W)	87.82 ± 1.42 (T)
Topological Sort	70.91 ± 3.42 (T)	88.20 ± 2.82 (T)
All Algorithms Concurrently	72.93 ± 1.62 (W)	88.69 ± 2.89 (T)

D Full Fine-tuning Performance

Table 4 presents the test performance under full fine-tuning. Interestingly, in this full fine-tuning setup, training on all algorithms jointly produced the best performance on *ogbg-molclintox*. This may indicate that when all layers are trainable, the model benefits from greater flexibility to overwrite biases introduced by less relevant pretraining tasks.

Table 4: Test performance on *ogbg-molhiv* and *ogbg-molclintox* with all layers trainable, reported as mean accuracy $\pm$ standard deviation over three runs. "W" denotes $\mu_{\text{model}} - \sigma_{\text{model}} > \mu_{\text{baseline}}$, "L" denotes $\mu_{\text{baseline}} - \sigma_{\text{baseline}} > \mu_{\text{model}}$, and "T" a tie otherwise.

Algorithm	**MolHIV Acc. (%)**	**MolClinTox Acc. (%)**
Baseline (No algorithms)	71.20 ± 0.68	86.85 ± 2.53
Activity Selector	71.55 ± 1.83 (T)	86.36 ± 6.04 (T)
Articulation Points	74.79 ± 1.08 (W)	89.95 ± 2.21 (W)
Bellman–Ford	73.52 ± 2.51 (T)	85.73 ± 1.19 (T)
BFS	$\mathbf{76.52 \pm 0.99}$ (W)	85.91 ± 0.94 (T)
Binary Search	73.49 ± 3.46 (T)	86.64 ± 1.74 (T)
Bridges	72.80 ± 0.52 (W)	85.16 ± 4.34 (T)
Bubble Sort	73.56 ± 1.05 (W)	87.38 ± 0.96 (T)
DAG Shortest Paths	73.71 ± 0.84 (W)	88.74 ± 1.89 (W)
Dijkstra	74.04 ± 2.21 (W)	87.07 ± 2.81 (T)
Find Maximum Subarray (Kadane)	72.12 ± 1.97 (T)	86.40 ± 2.90 (T)
Graham Scan	70.59 ± 3.30 (T)	87.69 ± 1.32 (T)
Insertion Sort	74.93 ± 0.94 (W)	87.48 ± 5.06 (T)
Jarvis' March	71.58 ± 2.57 (T)	89.10 ± 2.62 (T)
LCS Length	75.67 ± 1.60 (W)	87.82 ± 1.30 (T)
Matrix Chain Order	71.34 ± 1.69 (T)	89.55 ± 1.38 (W)
Minimum	71.91 ± 2.86 (T)	86.34 ± 1.29 (T)
MST Kruskal	72.36 ± 1.93 (T)	87.09 ± 1.33 (T)
MST Prim	75.05 ± 1.00 (W)	87.40 ± 3.19 (T)
Naive String Matcher	71.57 ± 0.39 (T)	87.38 ± 2.69 (T)
Optimal BST	72.29 ± 2.84 (T)	86.03 ± 0.55 (T)
Quicksort	72.15 ± 1.10 (T)	88.91 ± 1.24 (W)
Segments Intersect	69.18 ± 3.44 (L)	87.10 ± 1.46 (T)
Task Scheduling	73.91 ± 0.60 (W)	86.57 ± 4.46 (T)
Topological Sort	74.02 ± 0.12 (W)	85.88 ± 1.20 (T)
All Algorithms Concurrently	71.93 ± 1.59 (T)	$\mathbf{91.05 \pm 2.07}$ (W)