Availability Attacks on Graph Neural Networks

Shyam A. Tailor [1] **Miguel Tairum-Cruz** [2] **Tiago Azevedo** [2] **Nicholas D. Lane** [3 1] **Partha Maji** [2]

Abstract

Graph neural networks (GNNs) have become a popular approach for processing non-uniformly structured data in recent years. These models implement permutation-equivariant functions: their output does not depend on the order of the graph. Although reordering the graph does not affect model output, it is widely recognised that it may reduce inference latency. Less widely noted, however, is the observation that it is also possible to reorder the input graph to *increase* latency, representing a possible security (availability) vulnerability. Reordering attacks are difficult to mitigate, as finding an efficient processing order for an arbitrary graph is challenging, yet discovering an inefficient order is practically trivial in many cases: random shuffling is often sufficient. We focus on point cloud GNNs, which we find are especially susceptible to reordering attacks, and which may be deployed in real-time, safety-critical applications such as autonomous vehicles. We propose a lightweight reordering mechanism for spatial data, which can be used to mitigate reordering attacks in this special case. This mechanism is effective in defending against the slowdowns from shuffling, which we find for point cloud models can increase message propagation latency by 7.1×, with 81% increases to end-to-end latency with PosPool models at 1M points.

1. Introduction

In recent years graph neural networks (GNNs) have demonstrated excellent performance in applications ranging from drug discovery (Wu et al., 2017) to code analysis (Allamanis et al., 2017). Now that we begin to see GNNs being deployed more widely, practical concerns such as efficiency and security are paramount. These models may be deployed in safety-critical applications such as point cloud processing in autonomous vehicles, where even the smallest processing delay has safety implications. To attest the safety of such deployments, this paper asks the following question: can we attack the latency of GNN models?

There is a rich body of literature investigating attacks on neural networks (Szegedy et al., 2013; Shokri et al., 2017). An unpopular—but still critical—class of attacks are those that hinder the *availability* of a model (Shumailov et al., 2020; 2021; Hong et al., 2020). In this work, we attempt to increase a model's inference latency without necessarily attacking the model's confidentiality or integrity. By slowing inference, we can cause additional expense to the model deployment and, if the model is used for real-time applications, potential safety concerns.

GNNs are typically formulated using the message-passing paradigm, in which nodes send messages to each other and aggregate their received messages using a ***permutation-invariant*** function. The process of propagating messages around the graph is a sparse operation, which is typically memory-bound. It is necessary to minimize memory latency when processing the graph to reduce the propagation latency. One way to achieve this is to reorder the graph. Although there is research into accelerating graph processing by reordering the graph, it has not been noted explicitly that adversaries could (trivially) reorder the graph to slowdown processing. This is the key observation of our work. Defending against these reordering attacks is challenging: we typically only use a graph once when running inference, hence the cost of reordering as a defence must be cheap enough to be break-even in end-to-end latency. In this work, we propose a data reordering technique for point cloud data that is fast enough to be applied at inference time and provide robustness to reordering attacks. In summary, our work makes the following contributions:

1. We introduce data reordering attacks as an attack vector for GNNs. Unlike other attacks previously described on GNNs, this attack does not affect model integrity or confidentiality, but targets availability by affecting inference latency.

2. We show that reordering attacks are especially effective

[1]Department of Computer Science and Technology, University of Cambridge [2]ARM ML Research Lab [3]Samsung AI Center, Cambridge UK. Correspondence to: Shyam Tailor <sat62@cam.ac.uk>.

Proceedings of Adversarial Machine Learning Frontiers Workshop, ICML, Baltimore, Maryland, USA, 2022.

for point cloud data, with increases of 7.1× on message propagation latency. We observe increases to end-to-end model inference latency of 81%.

3. We propose a method for data reordering that leverages space-filling curves. Our approach is cheap enough to be applied at inference time as a defence against reordering attacks.

2. Background and Related Work

2.1. Graph Neural Networks

GNNs are usually described using the message-passing paradigm (Gilmer et al., 2017). In this paradigm, GNN layers define a message function that defines the message sent from node u to node v, an aggregation function that a node uses to combine all of its received messages, and an update function that a node applies to its previous representation and the aggregated messages to obtain an updated representation. Mathematically, this takes the form of: $\mathbf{h}_{l+1}^{(i)} = \text{Update}\left(\mathbf{h}_l^{(i)}, \text{Aggregate}_{j\in\mathcal{N}(i)}\left(\text{Message}\left(\mathbf{h}_l^{(i)}, \mathbf{h}_l^{(j)}\right)\right)\right)$, where $\mathbf{h}_l^{(i)}$ is the representation of node i at layer l, and $\mathcal{N}(i)$ defines the neighbourhood of node i.

Permutation Equivariance GNNs are *permutation equivariant*: therefore, for all permutation matrices $\mathbf{P} \in \mathbb{R}^{N\times N}$ we have $F(\mathbf{PAP}^\top, \mathbf{PX}) = \mathbf{P}F(\mathbf{A}, \mathbf{X})$, where $\mathbf{A}$ is an adjacency matrix for the graph and $\mathbf{X}$ is the matrix of node features. This property is why reordering attacks do not change model output: any labelling of a graph should yield an equivalent output.

2.1.1. Point Cloud GNNs

Point clouds are sets of points, typically in 3D space. There is no canonical ordering for a point cloud. PointNet (Qi et al., 2017a) is an early deep learning-based approach for operating on point clouds. The architecture consists of shared multi-layer perceptrons (MLPs) that operate on each point in parallel, followed by a global pooling operation to share information across the cloud. Recent GNN-based approaches build on PointNet by considering local aggregation: graphs are built from the point cloud using a grouping operation, such as radius or k-Nearest Neighbours (kNN).

Recent work by Liu et al. (2020), Li et al. (2021), and Tailor et al. (2021) demonstrate how to build point cloud layers that can be efficiently implemented. We focus on these layers in this work as they are orders of magnitude more efficient in practice (Tailor et al., 2021), enabling us to assess their performance at high point counts. In our experiments on point clouds, we consider SpMM as Tailor et al. (2021) demonstrated how to use this primitive to build point cloud GNNs. In addition, we consider PosPool, as proposed by Liu et al. (2020): $\mathbf{h}_{l+1}^{(i)} = \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} \Delta\mathbf{p}_{ij} \odot \mathbf{h}_l^{(j)}$, where $\Delta\mathbf{p}_{ij}$ is the vector $(\mathbf{p}_i - \mathbf{p}_j)$ repeated to match the length of $\mathbf{h}_l^{(j)}$, and $\odot$ is the Hadamard product. We note that point cloud GNNs may not be strictly permutation equivariant due to implementation details such as sampling operations; however, PosPool, as specified by the reference implementation, retains this property. Further discussion and experimental verification is supplied in Appendix A.2.

2.2. Attacks on Graph Neural Networks

In the wider literature, adversarial examples (Biggio et al., 2013; Szegedy et al., 2013), data poisoning attacks (Nelson et al., 2008; Jagielski et al., 2018), membership inference (Shokri et al., 2017; Salem et al., 2018), and training data extraction (Carlini et al., 2020) have been explored. With the rise of GNNs, there has been increased interest in building attacks for this class of models (Jin et al., 2020; Xu et al., 2019a; Geisler et al., 2021). Most attacks investigate how to make changes to the graph topology or the input features to either poison the data at training time or cause a misclassification at inference time. A growing body of literature aims to improve GNN model robustness to these attacks (Zhang & Zitnik, 2020; Tang et al., 2020; Entezari et al., 2020; Zhu et al., 2019). In addition to works targeting more general GNNs, there are works specifically targeting point cloud GNNs. Approaches including critical points removal (Qi et al., 2017a), point perturbations and introducing new points (Xiang et al., 2018; Zhao et al., 2020; Liu et al., 2019; Lang et al., 2020).

Unlike previous works attacking GNNs, our work does not attack model integrity or confidentiality: instead, we attack model *availability* i.e. preventing the timely and dependable access to information. It has been shown by Shumailov et al. (2020) that it is possible to find examples for language models that cause inference to increase by up to 3 orders of magnitude. Like this work, we demonstrate that providing adversarially modified input examples can be used to increase inference latency significantly. However, unlike Shumailov et al. (2020), we can find adversarial examples trivially by randomly permuting the graph.

2.3. Space Filling Curves

Space-filling curves (SFC) are defined as continuous functions with domain $[0, 1]$ and range that completely fills a d-dimensional unit hypercube. It is common to consider discrete variants of these curves: i.e. mappings using $\mathbb{N}_0$ rather than $\mathbb{R}$. We define a space-filling curve $\mathcal{C}$ as a bijective mapping $\mathcal{C} : \mathbb{N}_0^d \to \mathbb{N}_0$. We focus on Morton curves (Morton, 1966) due to their efficient implementation.

There is limited prior use of SFCs in the context of deep learning for point clouds. MortonNet (Thabet et al., 2019)

uses Morton curve orderings for self-supervised training. PointSCNet (Chen et al., 2022) uses the Morton curve to sample points, in conjunction with Farthest Point Sampling (FPS). Our work does not consider SFCs to improve modelling, but to defend against adversarial reordering attacks.

3. Methodology

3.1. Threat Model

In this work, we assume an adversary with the ability to provide an input graph to the target system, which then processes the input with a single CPU or GPU. We do not assume that the adversary can tamper with the system performing inference. However, we make no assumptions regarding the adversary's ability to tamper with pre-processing stages prior to inference. We also assume that the adversary has no access to the model parameters.

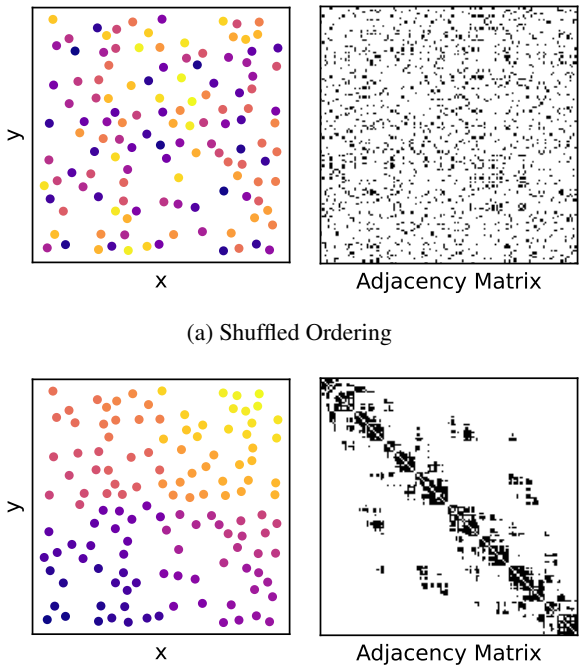
Our threat model cleanly captures both on-device inference and cloud inference scenarios. We argue that it also captures setups seen in autonomous vehicles. Our assumptions are weaker than those often made for adversarial attacks on point cloud models: we do not need to perturb any points and, as we demonstrate in the evaluation, we do not require a large time or compute budget to run our attack.

3.2. Attack Methods

We now present the data reordering attack against GNNs. We assume that we are given a graph $\mathcal{G}$ with node features $\mathbf{X} \in \mathbb{R}^{N \times F}$ where N is the number of nodes in the graph. Further, we may be given an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. Note that the adjacency matrix is not usually provided to the network in the context of point cloud data and is instead constructed dynamically during inference using the node features, which correspond to positional data in this case.

The adversary's objective is to slow inference. By destroying any underlying structure in the adjacency matrix, the adversary can cause the sparse operations to proceed significantly more slowly than before as optimizations made by the underlying hardware to hide memory latency become less effective. With high probability, the adversary can achieve this objective by generating a random permutation corresponding to $\{0, \ldots, N-1\}$ and calculating a reordered node feature matrix $\widetilde{\mathbf{X}}$. If present, the adjacency matrix can be similarly reordered to obtain $\widetilde{\mathbf{A}}$.

More Sophisticated Attacks Although we focus on random shuffling in this paper, we acknowledge that there are more sophisticated approaches that can further increase latency, at an increased cost to the attacker. These approaches could be derived by considering the myriad of reordering approaches in the literature (Balaji & Lucia, 2018), and inverting the objective: rather than finding an optimal reordering, instead search for an unfavourable ordering.



(a) Shuffled Ordering

(b) Morton Ordering

Figure 1. Visualization of how using an SFC to reorder points yields more hardware-friendly adjacency matrices after running a radius query. On the left, points are ordered randomly; we observe that there is no pattern to the corresponding adjacency matrix. On the right, we see that relabelling points according to the SFC yields a matrix that is clustered on the diagonal. This representation results in better locality, resulting in noticeable speedups.

3.3. Mitigating Ordering Attacks for Spatial Data

We now describe a defence measure for GNNs operating on spatial data, such as point clouds. Graph reordering is a challenging problem (Balaji & Lucia, 2018), and we leave solutions for the general case to future work; further discussion is provided in Appendix A.6. However, in the case of spatial data, we can apply SFCs to find an ordering for the nodes that preserves locality and therefore ensures that favourable structure is retained in the adjacency matrix.

Our method for spatial data ordering uses the following steps: **(1)** Discretize the space into a voxel grid. **(2)** Calculate the voxel (i, j, k) for each point. **(3)** Use the Morton curve to calculate a curve value $c = \mathcal{C}(i, j, k)$ for each point. **(4)** Sort the input points $\widetilde{\mathbf{X}}$ by the SFC values c to obtain $\bar{\mathbf{X}}$. If there are any features alongside the point positions, such as colour or intensity, they should also be sorted.

After performing these steps, inference can proceed as usual with $\bar{\mathbf{X}}$. This approach is simple and hardware efficient:

calculating the Morton value c requires simple bit manipulation operations. Unlike general reordering approaches, we have not needed to construct the graph using a grouping query, followed by an analysis on the resulting graph, which adds significant overhead (Appendix A.6). Instead, we are taking advantage of the properties of the space-filling curve—namely, the preservation of spatial locality when translating from 3D to 1D.

Figure 1 provides a visualization of our approach in 2D. At the top, we see points that have been randomly ordered: this is the scenario that an adversary could induce by using the attack described in the previous section. We see that the corresponding adjacency matrix has no clear pattern, which will result in poor locality, and, therefore, higher inference latency. By contrast, the bottom figure shows the result of labelling points according to their position on the SFC. In the adjacency matrix, we can see that the entries are clustered on the diagonals, with a small number of well-defined blocks off the diagonal, corresponding to discontinuities in the Morton curve. This ordering is far more amenable to efficient processing.

4. Evaluation

We will now show that point cloud GNNs are susceptible to shuffling attacks, and demonstrate the efficacy of the proposed SFC defence mechanism. Due to space limitations, we leave analysis for other data modalities to Appendix A.5. We run all experiments using an Intel i9-7900X for CPU, and an NVIDIA RTX8000 for GPU experiments. Further experiment details can be found in Appendix A.1.

4.1. Attacking Point Cloud Models

We will begin by assessing the risk to models operating on point cloud data. We will focus on the S3DIS dataset in this section as it is the largest popular dataset we can use without needing to downsample. Our evaluation will also include synthetic data to enable us to assess the performance of different operations at different scales.

We find that point cloud models—which have graph topologies with highly uniform degree distribution, and high mean node degree—are especially susceptible to reordering attacks. Even when we consider end-to-end model latency, we find that reordering attacks can induce noticeable slowdowns. We acknowledge that model deployers may process point clouds by sampling minibatches from the cloud, rather than processing the entire cloud at once. Our attack is viable even in this scenario: slowdown is induced at cloud sizes as small as 2^{14}, well within the sizes used by the literature (Hu et al., 2020). In addition, using larger minibatches is preferable as it often improves model performance (Wang et al., 2019; Xu et al., 2021; Hu et al., 2020).

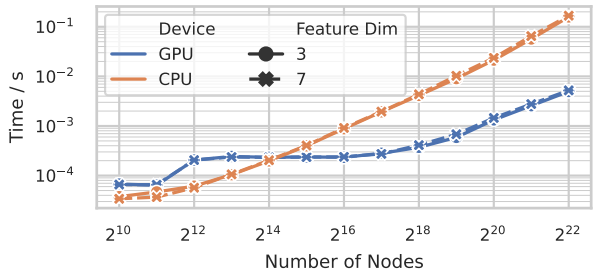


Figure 2. Attack latency, consisting of the time to calculate a new random permutation, followed by shuffling the point cloud.

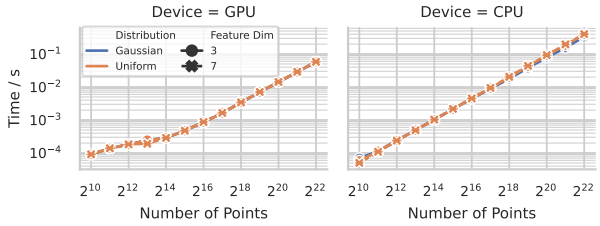


Figure 3. Time to perform SFC defence against shuffled points. Includes calculating SFC values, and sorting the feature matrix.

4.1.1. Cost of Reordering Points

To provide the reader with intuition on the total cost to an adversary to execute this attack, we provide latency figures for the time it takes an adversary to generate a random permutation and then apply it to the point cloud. Figure 2 illustrates the cost of this procedure for different problem sizes. We observe that this attack can be performed in time budgets of around 1ms, even for problem sizes reaching 2^{20}. We consider both raw 3D point, and points with 4 additional features, such as colour or intensity; we find that adding additional features has little impact on attack latency. Even this naive and unoptimized attack can be run in real-time: this attack latency can easily keep up with the rate points are generated by high-end LIDAR sensors, which is on the order of 1 million points per second (Vel). In comparison to other adversarial attacks, which require calculating perturbations to the input point cloud, this attack is far less computationally expensive.

4.1.2. Cost of Mounting a Space-Filling Curve Defence

Now that we have demonstrated that reordering attacks can be executed efficiently against point clouds, we now show that the cost of our proposed mitigation is sufficiently cheap to enable a robust defence without excessive overheads. We illustrate the cost of the defence in Figure 3. This procedure includes the cost of calculating the Morton curve value for

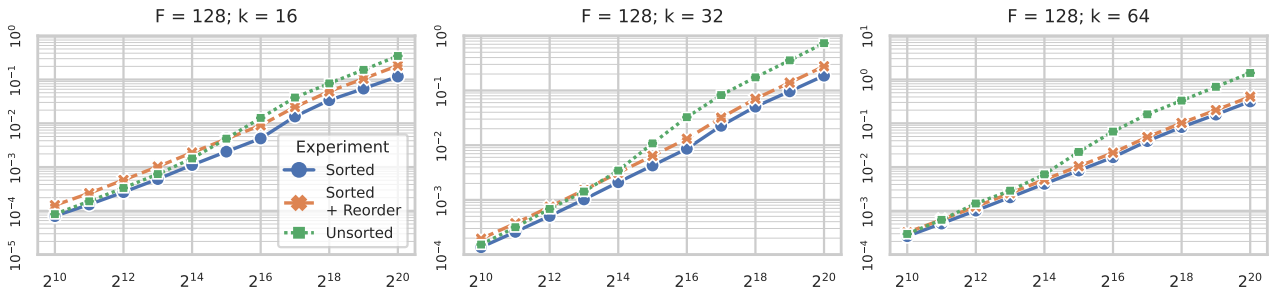


Figure 4. PosPool propagation latency when considering randomly generated uniform point clouds, with an associated feature dimension of 128 on CPU; extended results in Appendix A.3. y-axis is time in seconds.

each point, performing an arg-sort on the points by curve value, and then applying this new ordering to the model input. Unsurprisingly, this defence is more expensive than the shuffling attack, but, as we shall see in the next section, it is still sufficiently inexpensive that it is worth applying to untrusted data. We note that the cost of sorting may depend on the underlying data distribution: we performed these synthetic experiments with data generated from both the Normal and Uniform distributions. This represents a worst case with completely unsorted data: if the input data is partially sorted, then the cost of sorting may be reduced.

4.1.3. Impact on Message Propagation Time

We now look at the impact of reordering attacks on individual point cloud layers. In Figure 4, we provide the message propagation latency for PosPool on point cloud sizes ranging from 2^{10} to 2^{20} on CPU; figures for GPU and SpMM can be found in the Appendix. We consider constructing an adjacency matrix using kNN, with $k \in \{16, 32, 64\}$ using the point positions, and assume that each point has a feature vector of length F associated with it. This enables us to precisely assess the impact of neighbourhood size and feature dimensionality on message propagation latency. The points are generated using the uniform distribution; we consider using the unprocessed ordering and applying the Morton defence to the points. On CPU, we observe that the unsorted ordering is slower, even at low point counts, with large increases to latency occurring at $\geq 2^{14}$. Similar results are observed on GPU, with significant increases once the number of points exceeds 2^{15}. We also indicate the cost of implementing the SFC defence plus the propagation cost when using the sorted ordering; this corresponds to the worst-case cost of the defence, assuming it cannot be amortised across multiple layers. Even in this worst case, we observe that 2^{15} points often reaches the break-even point.

In Appendix A.3, we supply extended results where we consider feature dimensions in $\{64, 128, 256\}$. Increasing the feature dimension does not significantly affect the slowdown induced by reordering data. However, we find that increasing the number of neighbours per node (k) increases the slowdown ratio. For dimension 128, and point cloud size of 2^{20}, the slowdown ratio is $2.95\times$ / $3.91\times$ for $k = 16$; this increases to $4.48\times$ / $7.07\times$ for $k = 64$ on CPU / GPU.

We now proceed to investigate real-world data distributions. Table 1 provides the latency of processing point clouds from the S3DIS dataset on CPU; GPU numbers are in Appendix A.3. We consider three setups: firstly, when the data is processed using the ordering provided in the raw dataset; secondly, when the data is randomly shuffled, and thirdly with the data sorted using our SFC defence. As before, we use kNN to allow us to carefully control the number of neighbours each point has. We observe that the S3DIS data is also vulnerable to shuffling attack, as the unprocessed point clouds in the dataset are already spatially clustered: this is due to the pre-processing procedure used to generate the clouds (Armeni et al., 2017). Because this data is spatially clustered, shuffling the raw data causes significant increases in latency, with slowdown ratios of 2.6 / $3.2\times$ to 4.0 / $6.2\times$ on CPU / GPU. We also report the latency when the point clouds are sorted using our SFC defence, and observe that this ordering is better than the original ordering. This adds further evidence that our SFC defence is an effective method to achieve robustness to reordering attacks, even on real-world data distributions.

4.1.4. End-to-End Impact On Point Cloud Models

We now assess the impact of reordering attacks on end-to-end inference rather than individual layers. We implement the PosPool models described by Liu et al. (2020) and assess both classification and segmentation; for CPU experiments we halve the model hidden dimensions. Figure 5 plots the inference latency for a PosPool classifier as the point count varies from 2^{10} to 2^{20}; results for the segmentation variant can be found in Appendix A.4. For CPU, we observe increases in latency of 13.8% at 2^{14}, reaching 80.6% at 2^{20}.

Table 1. Cost of message propagation for a PosPool layer over point clouds taken from the S3DIS dataset on CPU; GPU results in Appendix A.3. Reordering attacks are effective on real data.

Feat Dim	k	Natural Order	Shuffled Order	Morton Order	Shuffled Ratio	Morton Ratio
64	**16**	5.77 ± 0.02	16.22 ± 0.01	$\mathbf{5.62 \pm 0.02}$	$2.81\times$	$0.97\times$
	32	9.11 ± 0.02	32.89 ± 0.05	$\mathbf{8.68 \pm 0.02}$	$3.61\times$	$0.95\times$
	64	15.83 ± 0.02	63.56 ± 0.09	$\mathbf{14.68 \pm 0.01}$	$4.01\times$	$0.92\times$
128	**16**	11.26 ± 0.02	28.92 ± 0.04	$\mathbf{10.87 \pm 0.02}$	$2.57\times$	$0.97\times$
	32	19.10 ± 0.02	60.56 ± 0.06	$\mathbf{17.76 \pm 0.02}$	$3.17\times$	$0.93\times$
	64	33.90 ± 0.02	116.70 ± 0.13	$\mathbf{30.44 \pm 0.02}$	$3.44\times$	$0.90\times$
256	**16**	21.03 ± 0.02	53.74 ± 0.023	$\mathbf{20.36 \pm 0.02}$	$2.56\times$	$0.97\times$
	32	38.61 ± 0.03	113.19 ± 0.04	$\mathbf{34.87 \pm 0.03}$	$2.93\times$	$0.90\times$
	64	86.44 ± 0.05	217.96 ± 0.06	$\mathbf{71.51 \pm 0.05}$	$2.52\times$	$0.83\times$

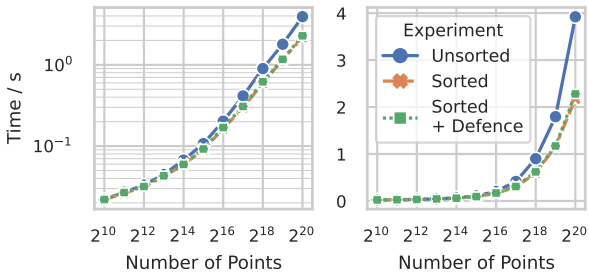


Figure 5. End-to-end latency for a PosPool classifier using point clouds from a Uniform distribution. Both logarithmic (left) and linear (right) y-axes are provided..

Table 2. Total latency for running a PosPool segmentation model over point clouds in the S3DIS dataset. Bad input data and sample ordering can induce major slowdowns on real data.

Data Ordering	Sample Ordering	Initial Defence Cost?	CPU / s	GPU / s
Shuffled	Unsorted	-	231.2 ± 0.3	35.36 ± 0.01
Original	Unsorted	-	143.8 ± 0.3	25.37 ± 0.01
Original	Morton	-	134.6 ± 0.2	24.47 ± 0.01
Morton	Morton	✓	131.7 ± 0.2	24.96 ± 0.01
Morton	Morton	✗	128.2 ± 0.2	23.74 ± 0.02

As expected from the experiments in the previous section, for GPU we see no significant differences in latency between unsorted and sorted data until 2^{16} points, where we observe an 8.0% increase in latency. This ratio quickly rises, with the increase at 2^{20} being 53.9%. When we include the cost of the defence, the increases are 4.5% and 46.3%, respectively.

Results on S3DIS We consider applying the full PosPool segmentation model to complete point clouds from S3DIS. The results are provided in Table 2. We observe that shuffling attacks are highly effective, causing a 80.4% / 48.9% increase in total CPU / GPU latency. We also observe that the Morton ordering is superior to the original data ordering. Once again, applying our defence mechanism can provide robustness to unfavourable input data distributions.

Naive Point Sampling Can Induce Slowdowns The model deployer may accidentally shuffle the graph. This may occur during the sampling step: point cloud models may downsample the point cloud. For example, the selected point indices are returned unsorted with random or furthest point sampling when using a naive implementation. In the case of grid subsampling, as used by PosPool, the sub-sampled voxel ordering may provide sub-optimal spatial clustering. The effect of naive sampling is illustrated in Table 2 with the original data ordering: we observe a noticeable 6.9% / 3.7% CPU / GPU slowdown if the sampled points are not post-processed to ensure that they are well-ordered (2nd / 3rd row).

5. Conclusion

The take-home message for practitioners is that data reordering attacks are trivial to perform, and can induce slowdowns on GNN models. Care must be taken when exposing GNNs to untrusted data. Our results demonstrate that these attacks are especially problematic for point cloud data: we observe attacks are viable from as few as 2^{14} points. We propose a defence mechanism for point cloud data, but finding a general-purpose solution which reaches the end-to-end break-even threshold on one-shot inference is a challenging problem. We acknowledge that our work has limitations: firstly, this attack is only viable when the graph is big enough. Secondly, we have focused on efficient GNN models in this paper; the impact on GNNs which need to explicitly calculate a message per-edge (e.g. GAT (Veličković et al., 2017), PNA (Corso et al., 2020)) will be considerably smaller, as the sparse operations are a smaller percentage of total runtime. Future work could assess more sophisticated attacks: random shuffling is an effective baseline, but is likely far from optimal.

Acknowledgements

Shyam Tailor is supported by EPSRC grants EP/M50659X/1 and EP/S001530/1 (the MOA project) and the European Research Council via the REDIAL project (Grant Agreement ID: 805194).

References

Velodyne HDL-32e. https://velodynelidar.com/products/hdl-32e/. Accessed: 28th July 2021.

Intel MKL. https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-documentation.html. Accessed: 16th May 2022.

Allamanis, M., Brockschmidt, M., and Khademi, M. Learning to represent programs with graphs. *arXiv preprint arXiv:1711.00740*, 2017.

Arai, J., Shiokawa, H., Yamamuro, T., Onizuka, M., and Iwamura, S. Rabbit Order: Just-in-Time Parallel Reordering for Fast Graph Analysis. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 22–31, May 2016. doi: 10.1109/IPDPS.2016.110. ISSN: 1530-2075.

Armeni, I., Sax, S., Zamir, A. R., and Savarese, S. Joint 2d-3d-semantic data for indoor scene understanding, 2017.

Balaji, V. and Lucia, B. When is graph reordering an optimization? studying the effect of lightweight graph reordering across applications and input graphs. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 203–214. IEEE, 2018.

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Š rndić, N., Laskov, P., Giacinto, G., and Roli, F. Evasion attacks against machine learning at test time. In *Advanced Information Systems Engineering*, pp. 387–402. Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-40994-3_25. URL https://doi.org/10.1007%2F978-3-642-40994-3_25.

Blanco, J. L. and Rai, P. K. nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees. https://github.com/jlblancoc/nanoflann, 2014.

Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., Oprea, A., and Raffel, C. Extracting training data from large language models, 2020. URL https://arxiv.org/abs/2012.07805.

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Cowan, M., Shen, H., Wang, L., Hu, Y., Ceze, L., Guestrin, C., and Krishnamurthy, A. Tvm: An automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, pp. 579–594, USA, 2018. USENIX Association. ISBN 9781931971478.

Chen, X., Wu, Y., Xu, W., Li, J., Dong, H., and Chen, Y. Pointscnet: Point cloud structure and correlation learning based on space filling curve-guided sampling. 2022. doi: 10.48550/ARXIV.2202.10251. URL https://arxiv.org/abs/2202.10251.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*, 2020.

Davis, T. A. and Hu, Y. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), dec 2011. ISSN 0098-3500. doi: 10.1145/2049662.2049663. URL https://doi.org/10.1145/2049662.2049663.

Derrow-Pinion, A., She, J., Wong, D., Lange, O., Hester, T., Perez, L., Nunkesser, M., Lee, S., Guo, X., Wiltshire, B., et al. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 3767–3776, 2021.

Entezari, N., Al-Sayouri, S. A., Darvishzadeh, A., and Papalexakis, E. E. *All You Need Is Low (Rank): Defending Against Adversarial Attacks on Graphs*, pp. 169–177. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450368223. URL https://doi.org/10.1145/3336191.3371789.

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., and Yin, D. Graph neural networks for social recommendation, 2019. URL https://arxiv.org/abs/1902.07243.

Fey, M. and Lenssen, J. E. Fast Graph Representation Learning with PyTorch Geometric, 5 2019. URL https://github.com/pyg-team/pytorch_geometric.

Geisler, S., Schmidt, T., Şirin, H., Zügner, D., Bojchevski, A., and Günnemann, S. Robustness of graph neural networks at scale, 2021. URL https://arxiv.org/abs/2110.14038.

Geng, T., Wu, C., Zhang, Y., Tan, C., Xie, C., You, H., Herbordt, M., Lin, Y., and Li, A. I-gcn: A graph convolutional network accelerator with runtime locality enhancement through islandization. In *MICRO-54: 54th Annual IEEE/ACM International Symposium*

on Microarchitecture, MICRO '21, pp. 1051–1063, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385572. doi: 10.1145/3466752.3480113. URL https://doi.org/10.1145/3466752.3480113.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017.

Hong, S., Kaya, Y., Modoranu, I.-V., and Dumitraş, T. A panda? no, it's a sloth: Slowdown attacks on adaptive multi-exit neural network inference. *arXiv preprint arXiv:2010.02432*, 2020.

Hu, Q., Yang, B., Xie, L., Rosa, S., Guo, Y., Wang, Z., Trigoni, N., and Markham, A. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11108–11117, 2020.

Jagielski, M., Oprea, A., Biggio, B., Liu, C., Nita-Rotaru, C., and Li, B. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning, 2018. URL https://arxiv.org/abs/1804.00308.

Jin, W., Li, Y., Xu, H., Wang, Y., Ji, S., Aggarwal, C., and Tang, J. Adversarial attacks and defenses on graphs: A review, a tool and empirical studies, 2020. URL https://arxiv.org/abs/2003.00653.

Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, 2017.

Lang, I., Kotlicki, U., and Avidan, S. Geometric adversarial attacks and defenses on 3d point clouds, 2020. URL https://arxiv.org/abs/2012.05657.

Leskovec, J. and Krevl, A. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

Li, Q., Han, Z., and Wu, X.-M. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pp. 3538–3545. Association for the Advancement of Artificial Intelligence, February 2018. ISBN 978-1-57735-800-8. URL https://www.research-collection.ethz.ch/handle/20.500.11850/368499. Accepted: 2019-10-07T09:59:39Z.

Li, Y., Chen, H., Cui, Z., Timofte, R., Pollefeys, M., Chirikjian, G., and Van Gool, L. Towards efficient graph convolutional networks for point cloud handling. *arXiv preprint arXiv:2104.05706*, 2021.

Liu, D., Yu, R., and Su, H. Adversarial shape perturbations on 3d point clouds, 2019. URL https://arxiv.org/abs/1908.06062.

Liu, Z., Hu, H., Cao, Y., Zhang, Z., and Tong, X. A closer look at local aggregation operators in point cloud analysis. *ECCV*, 2020.

Morton, G. M. A computer oriented geodetic data base and a new technique in file sequencing. 1966.

Nelson, B., Barreno, M., Chi, F. J., Joseph, A. D., Rubinstein, B. I. P., Saini, U., Sutton, C., Tygar, J. D., and Xia, K. Exploiting machine learning to subvert your spam filter. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, LEET'08, USA, 2008. USENIX Association.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017a.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017b.

Rusek, K., Suárez-Varela, J., Mestres, A., Barlet-Ros, P., and Cabellos-Aparicio, A. Unveiling the potential of graph neural networks for network modeling and optimization in sdn. In *Proceedings of the 2019 ACM Symposium on SDN Research*, pp. 140–151, 2019.

Salem, A., Zhang, Y., Humbert, M., Berrang, P., Fritz, M., and Backes, M. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models, 2018. URL https://arxiv.org/abs/1806.01246.

Shokri, R., Stronati, M., Song, C., and Shmatikov, V. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18, 2017. doi: 10.1109/SP.2017.41.

Shumailov, I., Zhao, Y., Bates, D., Papernot, N., Mullins, R., and Anderson, R. Sponge examples: Energy-latency attacks on neural networks, 2020. URL https://arxiv.org/abs/2006.03463.

Shumailov, I., Shumaylov, Z., Kazhdan, D., Zhao, Y., Papernot, N., Erdogdu, M. A., and Anderson, R. J. Manipulating sgd with data ordering attacks. *Advances in Neural Information Processing Systems*, 34:18021–18032, 2021.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks, 2013. URL https://arxiv.org/abs/1312.6199.

Tailor, S. A., de Jong, R., Azevedo, T., Mattina, M., and Maji, P. Towards efficient point cloud graph neural networks through architectural simplification, 2021. URL https://arxiv.org/abs/2108.06317.

Tailor, S. A., Opolka, F., Lio, P., and Lane, N. D. Do we need anistropic graph neural networks? In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=hl9ePdHO4_s.

Tang, X., Li, Y., Sun, Y., Yao, H., Mitra, P., and Wang, S. *Transferring Robustness for Graph Neural Network Against Poisoning Attacks*, pp. 600–608. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450368223. URL https://doi.org/10.1145/3336191.3371851.

Thabet, A., Alwassel, H., and Ghanem, B. Mortonnet: Self-supervised learning of local features in 3d point clouds, 2019. URL https://arxiv.org/abs/1904.00230.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38 (5):1–12, 2019.

Wang, Y., Feng, B., Li, G., Li, S., Deng, L., Xie, Y., and Ding, Y. GNNAdvisor: An adaptive and efficient runtime system for GNN acceleration on GPUs. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pp. 515–531. USENIX Association, July 2021. ISBN 978-1-939133-22-9. URL https://www.usenix.org/conference/osdi21/presentation/wang-yuke.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. 3d shapenets: A deep representation for volumetric shapes, 2015.

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. Moleculenet: A benchmark for molecular machine learning, 2017.

Xiang, C., Qi, C. R., and Li, B. Generating 3d adversarial point clouds, 2018. URL https://arxiv.org/abs/1809.07016.

Xu, H., Ma, Y., Liu, H., Deb, D., Liu, H., Tang, J., and Jain, A. K. Adversarial attacks and defenses in images, graphs and text: A review, 2019a. URL https://arxiv.org/abs/1909.08072.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How Powerful are Graph Neural Networks? *arXiv:1810.00826 [cs, stat]*, February 2019b. URL http://arxiv.org/abs/1810.00826. arXiv: 1810.00826.

Xu, M., Ding, R., Zhao, H., and Qi, X. Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3173–3182, 2021.

Xue, L. FRNN: Fixed Radius Nearest Neighbors Search Implemented In CUDA. https://github.com/lxxue/FRNN, 2020.

Yang, C., Buluc, A., and Owens, J. D. Design principles for sparse matrix multiplication on the gpu, 2018.

Zhang, G., He, H., and Katabi, D. Circuit-GNN: Graph neural networks for distributed circuit design. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7364–7373. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/zhang19e.html.

Zhang, J.-F. and Zhang, Z. Point-x: A spatial-locality-aware architecture for energy-efficient graph-based point-cloud deep learning. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, pp. 1078–1090, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385572. doi: 10.1145/3466752.3480081. URL https://doi.org/10.1145/3466752.3480081.

Zhang, X. and Zitnik, M. Gnnguard: Defending graph neural networks against adversarial attacks, 2020. URL https://arxiv.org/abs/2006.08149.

Zhao, Y., Shumailov, I., Mullins, R., and Anderson, R. Nudge attacks on point-cloud dnns, 2020. URL https://arxiv.org/abs/2011.11637.

Zhu, D., Zhang, Z., Cui, P., and Zhu, W. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '19, pp. 1399–1407, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330851. URL https://doi.org/10.1145/3292500.3330851.

A. Appendix

A.1. Experimental Details

Datasets We use Suitesparse (Davis & Hu, 2011) and SNAP (Leskovec & Krevl, 2014) datasets during our evaluation, along with S3DIS (Armeni et al., 2017). All of these datasets are freely available for academic use like this work.

Kernels We use highly optimised kernel implementations wherever possible. For SpMM, we use the Intel MKL (mkl) implementation on CPU. On GPU, we use an implementation based on Yang et al. (2018) provided by PyTorch Geometric (Fey & Lenssen, 2019). For PosPool, we had to write our own kernels. For CPU we used TVM (Chen et al., 2018) to implement optimised kernels; our baseline SpMM kernels built in TVM provided similar performance to Intel MKL. For GPU, we adapted the SpMM implementation provided by PyTorch Geometric, which is permitted under its MIT license.

PosPool Implementation We implemented our model using PyTorch, based on the specification provided in (Liu et al., 2020). To enable scaling to larger clouds, we adapt the methods used for grouping operations, such as those needed for constructing the graph, or upsampling. On CPU, we use nanoflann (Blanco & Rai, 2014) for fast kd-tree implementation of the required operations. On GPU, we use a brute-force implementation below 10,000 points, and FRNN (Xue, 2020) above this threshold. On CPU, we halve the model dimension throughout: i.e. instead of an initial hidden dimension of 144, we use 72.

We use the same preprocessing routines as the reference model. We adapt the radius and downsampling grid parameters relative to the reference model. We acknowledge that more significant changes may be made to the model in practice as the point cloud size changes, depending on precise details regarding the data distribution, such as density. However, the broad trends we observe regarding increases to latency on unsorted data will remain.

A.2. Permutation Equivariance in Point Cloud GNNs

Point cloud GNNs may not be strictly permutation equivariant depending on implementation choices. However, PosPool, as specified by the reference implementation, is strictly permutation equivariant. The grouping operation used by both our implementation and the reference is kNN, bounded by a fixed radius: i.e. any neighbours in the top-k which are outside of the specified radius are discarded. This is deterministic. Similarly, the subsampling operation used is voxel downsampling: in each voxel, the mean position of points falling into the voxel is returned. This is also deterministic. We verified this property for PosPool by assessing classification accuracy on ModelNet40 (Wu et al., 2015) while shuffling points. We obtained the exact same 92.301% test accuracy and 1.4699 test loss across 50 epochs of testing with different permutations, confirming that PosPool provides deterministic behaviour regardless of the input point permutation.

PointNet++ (Qi et al., 2017b), however, is not permutation equivariant (or deterministic), due to its choice of Furthest Point Sampling (FPS). Across 50 epochs using different permutations of the test set, and different random seeds for the FPS sampler, we observed test accuracy to be $91.96 \pm 0.19\%$, with minimum 91.61% and maximum 92.42%. As expected, there is a small difference between runs due to the loss of equivariance; however, the difference is sufficiently small that it can be neglected in many cases. We do not use PointNet++ in this paper regardless.

A.3. Extended Results on Propagation Latency

In Table 3 we provide details for message propagation latency for a PosPool layer using S3DIS data on GPU. In Figure 6 we provide extended results for message propagation latency for a PosPool layer on synthetic data. In Figure 7 we provide results for SpMM latency on synthetic data. Figure 8 and Figure 9 provide the analogous results for PosPool and SpMM respectively on GPU.

The trends are broadly consistent across devices and kernels. Between 2^{14} and 2^{16} we tend to see a large increase in the unsorted latency, relative to the Morton sorted points. The increase in latency can be substantial, with increases exceeding $5\times$ frequently occurring.

A.4. Extended End-To-End Model Results

Figure 10 provides end-to-end latency results on a PosPool segmentation model. Trends are similar to those seen for classification models, albeit less severe as a percentage of total model runtime because the upsampling layers account for a

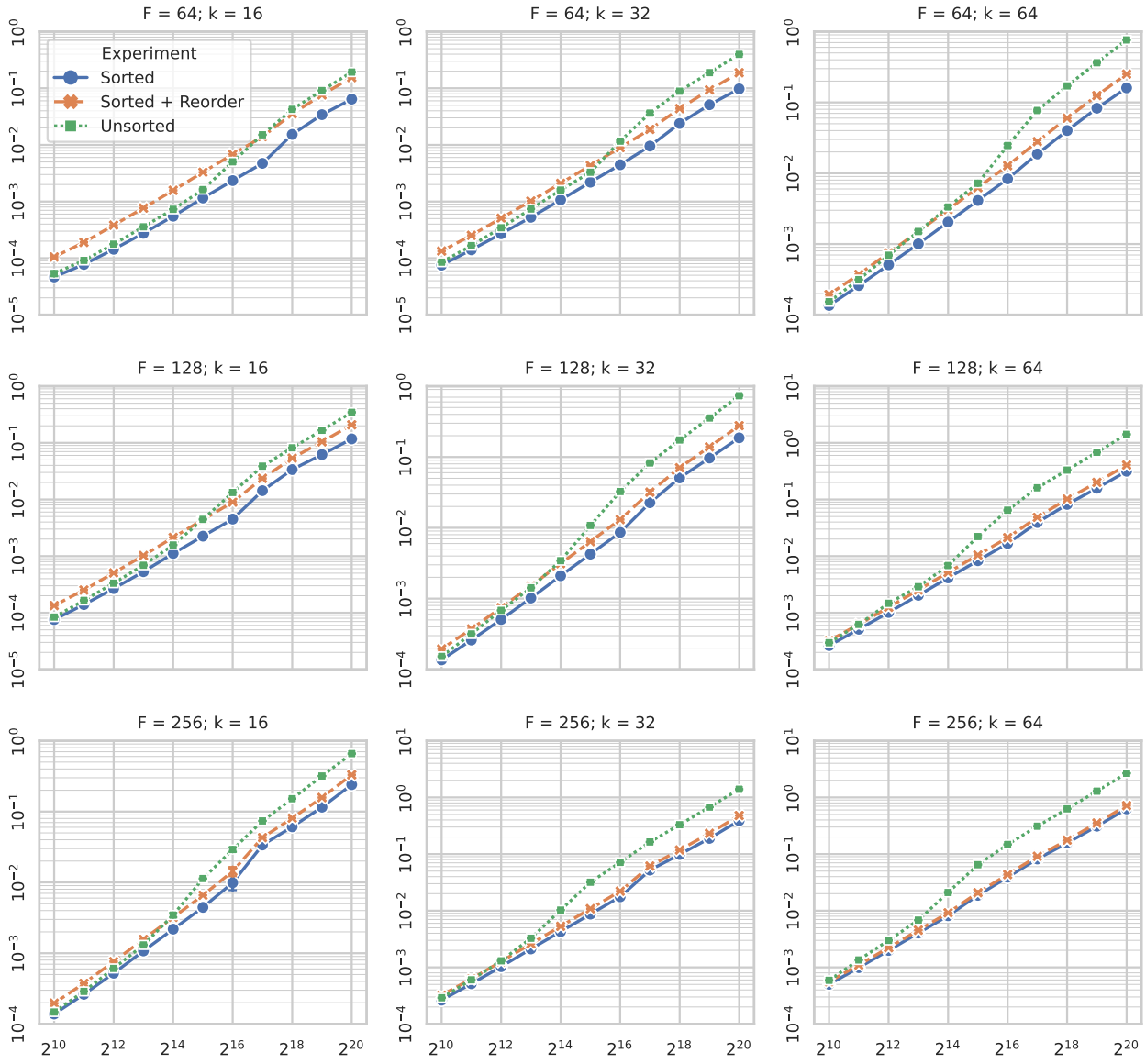


Figure 6. Extended PosPool propagation latency results for synthetic data on CPU.

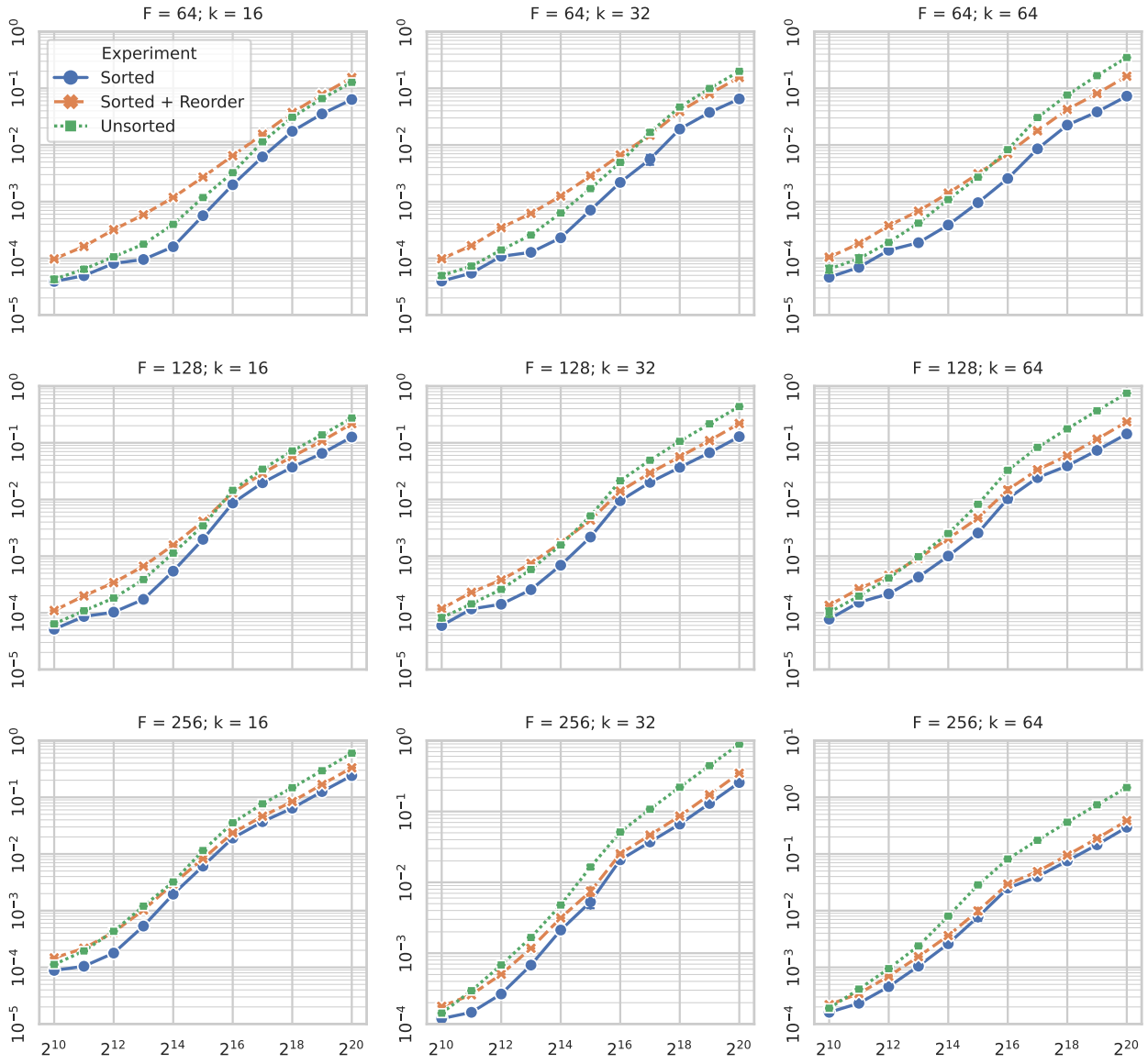


Figure 7. SpMM propagation latency results for synthetic data on CPU.

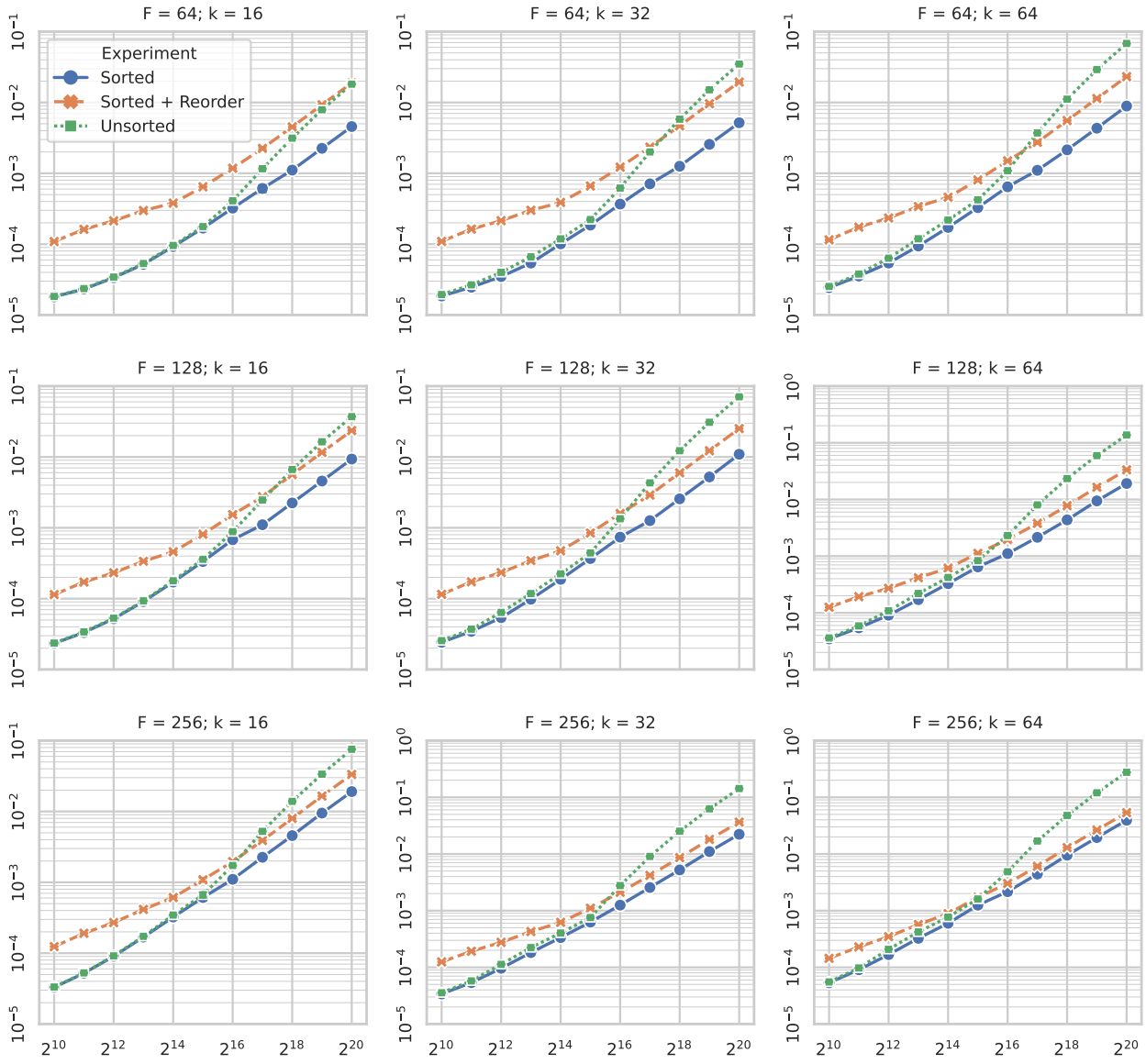


Figure 8. PosPool propagation latency results for synthetic data on GPU.

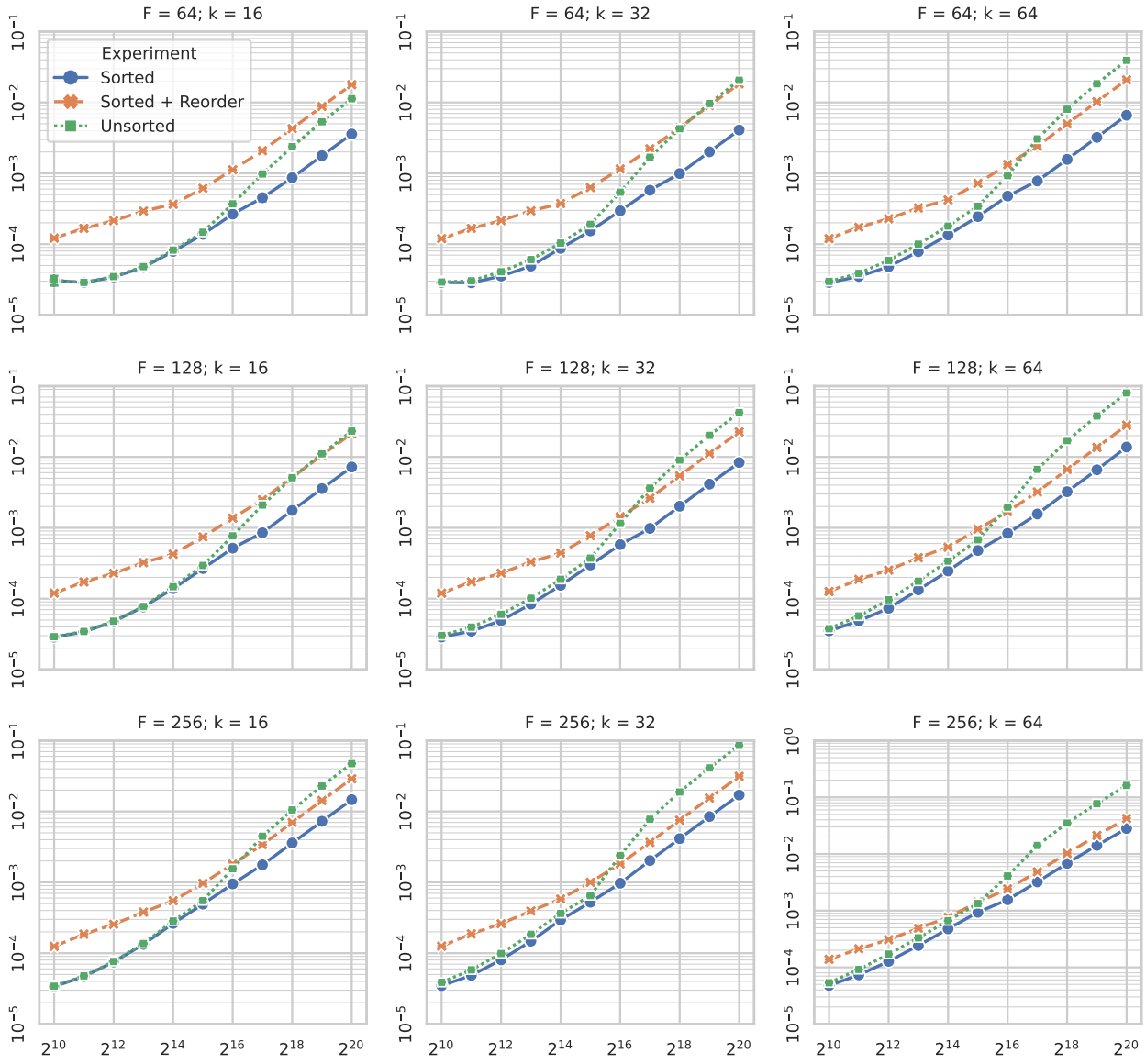
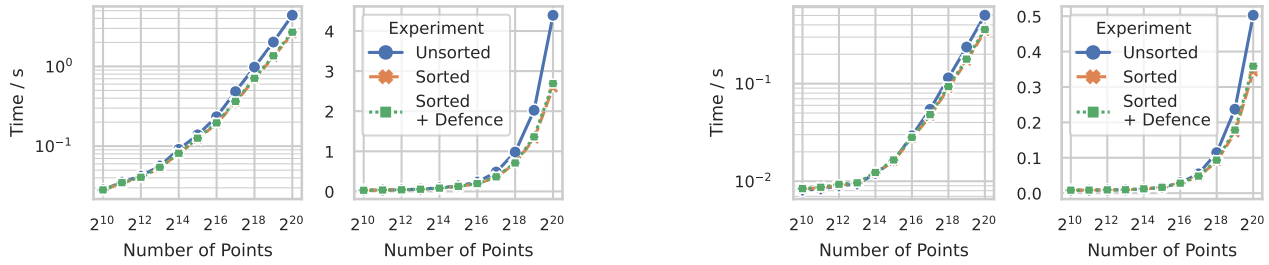


Figure 9. SpMM propagation latency results for synthetic data on GPU.

Table 3. Total cost of message propagation for a PosPool layer over unmodified point clouds taken from the S3DIS dataset. Run on GPU.

Feat Dim	k	Natural Order	Shuffled Order	Morton Order	Shuffled Ratio	Morton Ratio
64	16	**0.420 ± 0.002**	1.457 ± 0.000	0.421 ± 0.001	3.47×	1.00×
	32	0.510 ± 0.001	2.824 ± 0.001	**0.503 ± 0.001**	5.54×	0.99×
	64	0.898 ± 0.001	5.541 ± 0.002	**0.875 ± 0.002**	6.17×	0.97×
128	16	0.847 ± 0.002	2.964 ± 0.002	**0.845 ± 0.001**	3.50×	1.00×
	32	1.024 ± 0.002	5.700 ± 0.002	**1.009 ± 0.001**	5.57×	0.99×
	64	1.802 ± 0.004	11.153 ± 0.004	**1.758 ± 0.003**	6.19×	0.98×
256	16	1.700 ± 0.003	6.026 ± 0.002	**1.699 ± 0.002**	3.54×	1.00×
	32	2.062 ± 0.003	11.502 ± 0.004	**2.027 ± 0.002**	5.58×	0.98×
	64	3.616 ± 0.006	22.407 ± 0.007	**3.539 ± 0.004**	6.20×	0.98×



(a) CPU (b) CUDA

Figure 10. End-to-end inference latency for a PosPool segmentation model.

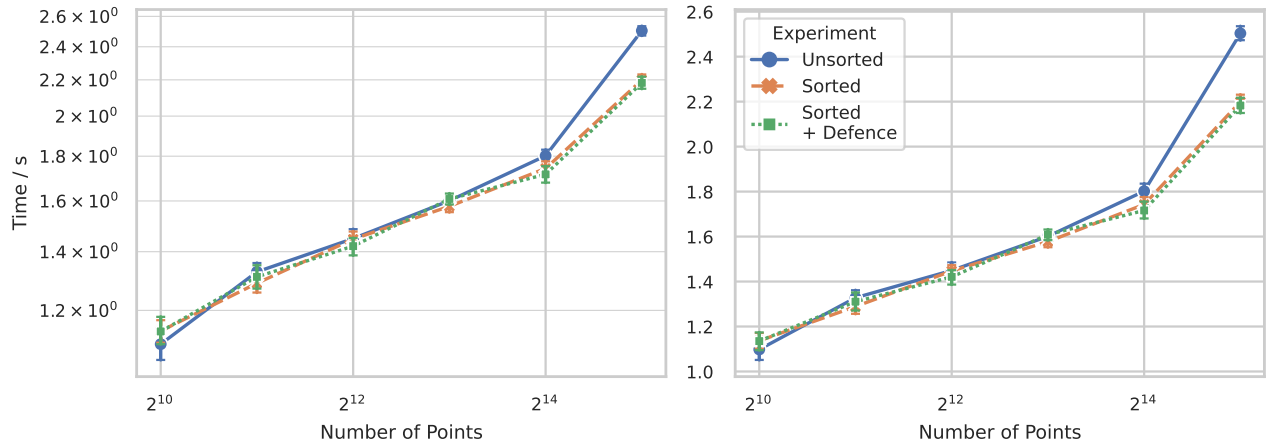
non-trivial proportion of model runtime.

A.4.1. Raspberry Pi Experiments

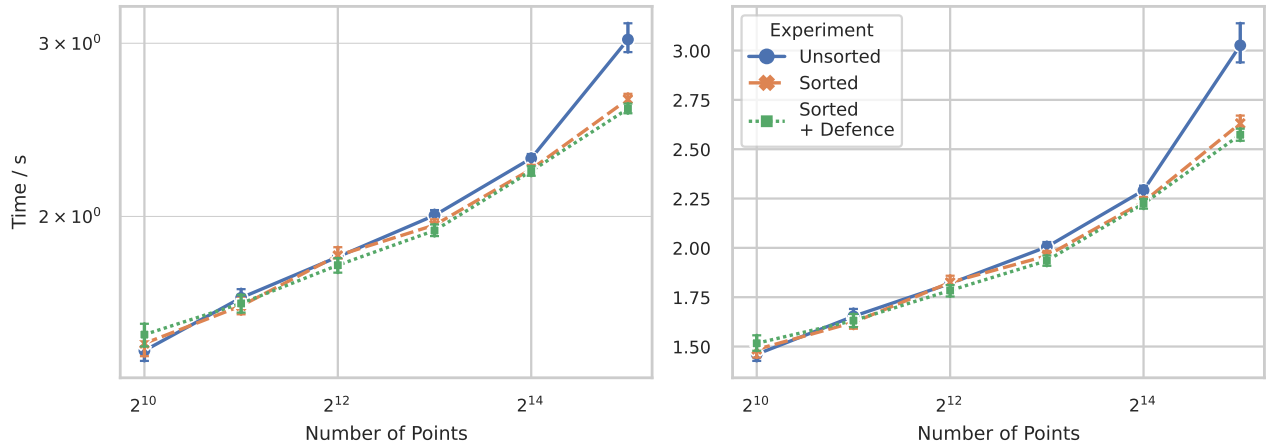
We also ran end-to-end experiments on a Raspberry Pi Model 4B in order to demonstrate that the lower end devices with significantly more limited cache (1MB LLC in the case of the Raspberry Pi) are also affected. Latency numbers are provided in Figure 11. We observe noticeable deviations above 2^{14} points, as we see with our experiments on higher-end CPUs and GPUs.

Table 4. The impact of random shuffling on inference latency in a GCN layer with dimension 128 across a variety of graphs taken from SNAP (Leskovec & Krevl, 2014) and SuiteSparse (Davis & Hu, 2011).

Dataset	Operations	CPU / ms Unshuffled	CPU / ms Shuffled	Slowdown	GPU / ms Unshuffled	GPU / ms Shuffled	Slowdown
web-Google	**Sparse Only**	198.4 ± 1.0	198.5 ± 2.1	1.00×	14.85 ± 0.80	14.08 ± 0.05	0.95×
	Full Layer	256.0 ± 1.0	255.0 ± 1.0	1.00×	16.79 ± 0.24	16.58 ± 0.09	0.99×
soc-pokec	**Sparse Only**	509.8 ± 5.0	532.5 ± 2.6	1.04×	42.45 ± 0.03	49.26 ± 0.04	1.16×
	Full Layer	600.8 ± 7.9	629.0 ± 3.7	1.05×	46.74 ± 0.15	53.66 ± 0.04	1.15×
roadnet-ca	**Sparse Only**	340.9 ± 1.5	384.6 ± 1.3	1.13×	20.67 ± 0.22	22.28 ± 0.21	1.08×
	Full Layer	477.4 ± 1.8	521.5 ± 2.2	1.09×	26.78 ± 0.47	28.38 ± 0.38	1.06×
as-skitter	**Sparse Only**	418.6 ± 1.5	456.5 ± 7.9	1.09×	27.14 ± 0.17	37.86 ± 0.04	1.39×
	Full Layer	535.8 ± 1.9	560.6 ± 19.0	1.05×	32.12 ± 0.20	41.78 ± 0.13	1.30×
freescale1	**Sparse Only**	596.3 ± 2.5	655.9 ± 3.5	1.10×	38.35 ± 0.28	42.74 ± 0.81	1.11×
	Full Layer	791.4 ± 5.4	851.8 ± 3.6	1.08×	49.98 ± 1.38	53.27 ± 1.03	1.07×



(a) Classifier



(b) Segmentation

Figure 11. End-to-end inference latency for PosPool models run on a Raspberry Pi Model 4B.

A.5. Reordering Attacks on Graph Data

To demonstrate that reordering attacks generalise beyond point clouds, we now consider the impact of reordering attacks across a variety of graph topologies with a general purpose GNN layer. Table 4 provides the inference latency for a GCN layer (Kipf & Welling, 2017) operating on unshuffled and shuffled data. We include graphs from Stanford SNAP (Leskovec & Krevl, 2014) and SuiteSparse (Davis & Hu, 2011), and provide timings for the entire layer, and just the sparse graph-level operations. Domains covered include road networks, social graphs, computer network topologies, and circuit simulations. We use these datasets to represent domains to which GNNs have been applied in the literature (Derrow-Pinion et al., 2021; Fan et al., 2019; Rusek et al., 2019; Zhang et al., 2019). We observe consistent trends on many datasets that the propagation latency increases on the shuffled graph relative to the original graph ordering. We note that the unshuffled ordering is sub-optimal, limiting the impact of reordering attacks: applying reordering can improve latency by >20%, as we show in Appendix A.6. Even so, this simplistic attack is still capable of inducing measureable slowdowns. We note that random shuffling attacks are not universally successful: a more sophisticated method is required for scale-free graphs, such as web graphs. However, for `as-Skitter` we observe increases to inference latency of up to 39% for the sparse operations. Although we focus on GCN, our analysis is valid for other GNN layers, including GIN (Xu et al., 2019b), GraphSAGE (Hamilton et al., 2017), and EGC (Tailor et al., 2022). These layers also rely on the same SpMM primitive used by GCN for graph-level operations, which constitutes most of the inference latency, as seen in Table 4. In the appendix, we

Table 5. Latency for permuting adjacency matrices. Shuffling the adjacency matrix is sufficiently cheap that real time attacks are viable.

Dataset	CPU / ms	GPU / ms
web-Google	12.1 ± 0.2	0.53 ± 0.00
soc-pokec	80.5 ± 0.9	3.74 ± 0.01
roadnet-ca	13.5 ± 1.3	0.40 ± 0.01
as-skitter	52.6 ± 0.3	1.58 ± 0.00
freescale1	41.1 ± 1.2	1.22 ± 0.01

Table 6. The impact of random shuffling on inference latency in a GCN layer with dimension 64 and 256; extended from Table 4. Measurements are for sparse components of the GCN layer.

		CPU / ms			GPU / ms		
Dataset	**Feat Dim**	**Unshuffled**	**Shuffled**	**Slowdown**	**Unshuffled**	**Shuffled**	**Slowdown**
web-Google	**64**	103.08 ± 0.93	97.09 ± 1.15	$0.94\times$	7.61 ± 0.02	7.55 ± 0.24	$0.99\times$
	256	455.6 ± 7.47	454.12 ± 7.44	$1.00\times$	32.3 ± 0.26	32.44 ± 0.32	$1.00\times$
soc-pokec	**64**	234.36 ± 1.36	247.35 ± 0.9	$1.06\times$	20.59 ± 0.13	24.49 ± 0.34	$1.19\times$
	256	1161.01 ± 4.55	1224.94 ± 3.01	$1.06\times$	93.53 ± 0.16	106.3 ± 0.07	$1.14\times$
roadnet-ca	**64**	156.03 ± 3.0	190.13 ± 2.8	$1.22\times$	11.25 ± 0.42	11.74 ± 0.42	$1.04\times$
	256	660.05 ± 4.31	780.6 ± 3.02	$1.18\times$	52.37 ± 0.83	55.66 ± 0.4	$1.06\times$
as-skitter	**64**	183.12 ± 0.95	205.54 ± 1.01	$1.12\times$	13.21 ± 0.53	19.42 ± 0.23	$1.47\times$
	256	886.13 ± 7.98	985.53 ± 3.21	$1.11\times$	65.4 ± 1.34	82.63 ± 0.29	$1.26\times$
freescale1	**64**	322.13 ± 1.39	341.76 ± 1.34	$1.06\times$	17.93 ± 0.11	20.13 ± 0.45	$1.12\times$
	256	1204.4 ± 3.29	1400.69 ± 3.35	$1.16\times$	98.18 ± 0.87	105.13 ± 0.42	$1.07\times$

extend our analysis for GCN layers with different feature dimensions in Table 6. We observe no major changes to the trends.

We also provide the latency to permute edges in Table 5. Shuffling the adjacency matrix is also sufficiently cheap that real-time attacks are viable for attackers. However, defensive graph reordering approaches must also consider the cost of finding a permutation, which may cost too much to break-even in an inference scenario where we use the graph once. Lightweight approaches such as Rabbit ordering (Arai et al., 2016) may be a viable defence strategy on CPU, but we do not observe it to be universally effective, and it is substantially slower than our SFC approach. Fortunately, we note that hardware support for graph reordering in GNN accelerators is being developed (Geng et al., 2021; Zhang & Zhang, 2021), which can help mitigate these attacks. For more details, see Appendix A.6

A.6. General Reordering Costs

We report extended results for the efficacy of reordering attacks across different graph topologies in Table 6; this is an extension of Table 4. We observe similar trends to before, although the maximum effect sizes may be larger: for example, we see increases of 22% on `roadnet-ca` on CPU, and 47% on `as-skitter` on GPU.

For completeness, we consider general graph reordering strategies, and their applicability as a defence strategy against reordering attacks. Data for Rabbit ordering (Arai et al., 2016), a lightweight reordering strategy that may be applied just-in-time, in provided in Table 7. We see that on CPU, it may be possible to use Rabbit ordering as a viable defence strategy on some, but not all, datasets. The break-even cost depends on factors such as model depth and the benefit provided by new ordering. However, there is limited evidence that this strategy will be effective on GPU: we are not aware of any reordering strategy implemented on GPU, although we acknowledge that simple degree-sorting (Balaji & Lucia, 2018) could be applied in principle. The reader should note that degree-sorting will be ineffective on point cloud data, as the degree distribution is not skewed. In practice, Rabbit ordering has relatively high synchronisation overheads, limiting its efficiency on GPUs; GNNAdvisor (Wang et al., 2021) runs Rabbit ordering on CPU, and applies the generated permutation on GPU.

We also provide timings for running Rabbit ordering on synthetic point clouds, after using kNN queries to construct the necessary graph. In the worst case, our reordering approach requires 92.4 ± 0.6 ms to calculate a permutation for the same

Table 7. Latency associated with calculating and applying Rabbit Order. The efficacy of this approach depends on (1) the time required to calculate the ordering; (2) the time required to apply the ordering; (3) the improvement offered by the new ordering relative to unshuffled or shuffled results; (4) the depth of the GNN, which is frequently limited to just 3 or 4 layers due to the oversmoothing phenomenon (Li et al., 2018). For example, this approach will likely be ineffective on `soc-pokec`.

Dataset	Unshuffled Order / ms	Shuffled Order / ms	Rabbit Order / ms	Time to Apply Rabbit Order / ms	Time to Calculate Rabbit Order / ms
web-Google	198.4 ± 1.0	198.5 ± 2.1	140.9 ± 7.3	5.5 ± 0.0	72.1
soc-pokec	509.8 ± 5.0	532.5 ± 2.6	478.5 ± 11.0	52.7 ± 8.0	471.0
roadnet-ca	340.9 ± 1.5	384.6 ± 1.3	314.1 ± 1.1	5.0 ± 0.0	114.7
as-skitter	418.6 ± 1.5	456.5 ± 7.9	331.5 ± 2.0	43.0 ± 0.1	151.7
freescale1	596.3 ± 2.5	655.9 ± 3.5	495.3 ± 2.9	16.4 ± 0.1	291.5
Point Cloud ($2^{20}, k = 16$)	-	-	-	-	415.7
Point Cloud ($2^{20}, k = 32$)	-	-	-	-	558.8
Point Cloud ($2^{20}, k = 64$)	-	-	-	-	802.7

size point cloud. This is at least a $4.5\times$ improvement relative to Rabbit.

Finally, we note that graph reordering approaches are now being implemented in hardware (Zhang & Zhang, 2021; Geng et al., 2021). If these approaches become widespread, then this can mitigate the impacts of this attack, since the graph will be reordered regardless to optimise for performance.