Short-Range Oversquashing

Yaaqov Mishayev
Technion – Israel Institute of Technology
yakov-m@campus.technion.ac.il

Yonatan Sverdlov
Technion – Israel Institute of Technology
yonatans@campus.technion.ac.il

Tal Amir
Technion – Israel Institute of Technology
talamir@technion.ac.il

Nadav Dym
Technion – Israel Institute of Technology
nadavdym@technion.ac.il

Abstract

Message Passing Neural Networks (MPNNs) are widely used for learning on graphs, but their ability to process long-range information is limited by the phenomenon of *oversquashing*. This limitation has led some researchers to advocate Graph Transformers as a better alternative, whereas others suggest that it can be mitigated within the MPNN framework, using virtual nodes or other rewiring techniques.

In this work, we demonstrate that oversquashing is not limited to long-range tasks, but can also arise in short-range problems. This observation allows us to disentangle two distinct mechanisms underlying oversquashing: (1) the *bottleneck phenomenon*, which can arise even in low-range settings, and (2) the *vanishing gradient* phenomenon, which is closely associated with long-range tasks.

We further show that the short-range bottleneck effect is not captured by existing explanations for oversquashing, and that adding virtual nodes does not resolve it. In contrast, transformers do succeed in such tasks, positioning them as the more compelling solution to oversquashing, compared to specialized MPNNs.[1]

1 Introduction

Graph Neural Networks (GNNs) are the leading tool for learning on graph-structured data, with many of the most popular models falling into the category of *Message Passing Neural Networks* (MPNNs). While MPNNs are computationally very efficient due to their ability to leverage graph sparsity, they are known to be successful only when using a small number of MPNN layers, typically 2–4. The difficulty in training deep MPNNs is commonly attributed to the phenomena of *oversmoothing* [1] and *oversquashing* [2]. Oversmoothing is the phenomenon in which, as the number of MPNN layers increases, node features become nearly indistinguishable from one another, with the extreme case often termed *total collapse*. Our focus in this paper will primarily be on oversquashing.

The term *oversquashing*, coined by Alon and Yahav [2], refers to the difficulty of training MPNNs on *long-range tasks*, that is, tasks that require communication between distant nodes to solve the problem accurately. The authors explained that this difficulty is caused by a *bottleneck effect*, where the intermediate nodes on the path between two distant nodes need to have an increasingly large feature dimension in order to solve the problem by message passing. Later papers put more emphasis on low graph connectivity and vanishing gradients as the essential component leading to oversquashing.

Oversmoothing and oversquashing are at the center of active discussion in the graph-learning community. On the one hand, many papers attempt to enable the use of MPNNs for long-range learning using techniques such as virtual nodes and other rewiring methods, which are aimed at reducing the range and difficulty of the learning problem. In a different direction, several papers have argued that *Graph Transformers* (GTs) outperform MPNNs due to their ability to handle long-range tasks [3–6].

[1] Code available at https://github.com/YakovM93/Short-Range-Oversquashing.

Mishayev et al., Short-Range Oversquashing. *Proceedings of the Fourth Learning on Graphs Conference (LoG 2025)*, PMLR 269, Hybrid Event, December 10–12, 2025.

Later results cast doubt on this claim, showing that with careful training, MPNNs with virtual nodes can obtain competitive results in many graph benchmarks, including Long Range Graph Benchmark (LRGB) [6], which consists of tasks that arguably require long-range interactions. Similarly, it was argued in a recent work [7] that MPNNs with virtual nodes are able to simulate the attention mechanism: "despite recent efforts, we still lack good benchmark datasets where GT can outperform MPNN by a large margin."

Our main goal in this paper is to advance the theoretical understanding of oversquashing by demonstrating that it can also arise in short-range tasks. Specifically, we construct a family of graph-learning problems that admit exact solutions with just two MPNN iterations, yet we prove that any MPNN must employ very large node-feature dimensions to solve them. Thus, these problems are affected by the bottleneck effect, even though the underlying graphs are well connected. We also show empirically that the vanishing gradient problem does not occur for these problems.

In contrast, we show that popular synthetic long-range tasks considered in the literature suffer from vanishing gradients but not from the bottleneck effect. Thus, we claim that these are two distinct effects that were inadvertently mixed together.

In addition, we show empirically that standard MPNNs, even when augmented with virtual nodes, perform poorly on these problems, whereas transformers solve them easily. This yields an interpretable test case in which MPNNs with virtual nodes are clearly outperformed by transformers.

1.1 Related Work

Oversquashing in graph neural networks. The oversquashing phenomenon was first identified by Alon and Yahav [2], who demonstrated that MPNNs struggle to propagate information between distant nodes due to an exponential growth in the nodes' receptive field. They introduced the Tree Neighbors-Match problem, discussed in section 2, as a canonical example, where the bottleneck effect arises as the problem radius increases, arguing that the exponential growth of receptive fields with depth creates information bottlenecks at intermediate nodes.

Theoretical explanations of oversquashing. Several theoretical frameworks have been proposed to explain oversquashing. Topping et al. [8] connected oversquashing to the Ricci curvature of graph edges, proving that edges with negative curvature act as information bottlenecks and deriving bounds on gradient norms in terms of curvature. Di Giovanni et al. [9] provided direct gradient-decay analysis, deriving bounds on Jacobian norms that predict exponential decay when either the distance between nodes or the number of message-passing iterations is large.

Spectral properties and graph connectivity. The *spectral gap* (smallest nonzero eigenvalue λ_1 of the normalized graph Laplacian) has been proposed as a key indicator of oversquashing potential. When λ_1 is close to zero, the graph is nearly disconnected into multiple components, suggesting poor information flow. Several works have used spectral graph theory to predict and mitigate oversquashing, proposing rewiring strategies that increase λ_1 to improve information propagation Topping et al. [8]. Black et al. [10] introduced *effective resistance* as a measure for predicting oversquashing. For nodes u and v, the effective resistance is $R_{u,v} = (1_u - 1_v)L^+(1_u - 1_v)$, where L^+ is the pseudoinverse of the graph Laplacian. This quantity, borrowed from electrical network theory, accounts for all paths between nodes and is proportional to commute time in random walks [11]. Black et al. proved upper bounds on gradient norms in terms of effective resistance, establishing connections between high resistance and vanishing gradients.

Graph Transformers and attention mechanisms. Because self attention enables all-pairs communication in a single hop, Graph Transformers are often argued to mitigate oversquashing when paired with structural/positional encodings (e.g., Graphormer [3]; GPS [4]; Exphormer [5]). Cai et al. [7] showed theoretically that MPNN with virtual nodes (VN) can approximate self attention (including linear transformers) and, empirically, that strong MPNN+VN baselines are competitive on LRGB, sharpening the GT–MPNN comparison. Rosenbluth et al. [12] studied uniform expressivity and proved that GT and MPNN+VN are incomparable—neither subsumes the other—while much of "universality" in non-uniform settings stems from powerful positional encodings; their experiments report mixed outcomes across datasets. The Long-Range Graph Benchmark (LRGB) [6] was introduced to stress long-distance interactions, and subsequent re-evaluations with stronger baselines have narrowed parts of the once-reported transformer advantage [13].

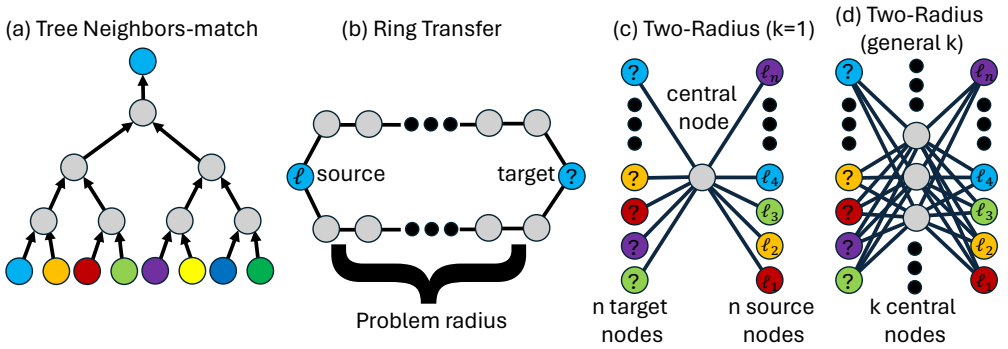


Figure 1: Illustration of synthetic graph-transfer problems. (a) **Tree Neighbors-Match**: information is transferred from leaves to a target node through a tree of depth r. (b) **Ring Transfer**: a source and target are connected by two disjoint paths of length r. (c) **Two-Radius**: n sources, n targets, and a single central node. (d) **Generalized Two-Radius**: k central nodes. Node colors represent source and target identifiers; gray denotes central nodes.

Oversmoothing and related phenomena. Oversmoothing is another fundamental limitation of deep GNNs, in which node features become indistinguishable as the number of layers increases. Rusch et al. [1] provide a comprehensive survey of oversmoothing, showing that it emerges as a consequence of repeated averaging operations in deep networks. Oversmoothing is inherently a deep-network phenomenon, requiring many layers before node features converge to similar values.

Information-theoretic perspectives. Alon and Yahav [2] introduced information-theoretic arguments to understand oversquashing, connecting information capacity requirements to exponentially growing receptive fields in deep networks. They argued that intermediate nodes must store information about an exponentially growing neighborhood, creating fundamental bottlenecks. This perspective has influenced subsequent work on understanding the theoretical limits of message-passing architectures and motivated the search for alternative architectures that avoid these bottlenecks.

More recently, Arnaiz-Rodriguez and Errica [14] published a broad position paper that critically examines common beliefs in graph machine learning, exposing conceptual ambiguities surrounding notions such as oversmoothing and oversquashing. Their work argues that many of these ideas have become intertwined in the literature and calls for clearer distinctions between them. This perspective complements ours by emphasizing the importance of separating computational bottlenecks (oversquashing) from topological assumptions, thereby motivating the more fine-grained theoretical and empirical analysis we undertake here.

1.2 Notation and Preliminaries

We begin by introducing notation. Graphs are denoted by $G = (V, E, \mathbf{X})$, where V is a finite set of nodes, E is the set of graph edges, and $\mathbf{X} = (x_v)_{v \in V}$ denotes node feature vectors $x_v \in \mathbb{R}^d$. The set of neighbors of node v is denoted by $\mathcal{N}_v$. Message-Passing Neural Networks (MPNNs) are graph neural networks that update each node's feature by combining its own feature with the features of its neighbors. Namely, the feature vector h_v^k at each layer k is iteratively computed by

$$h_v^0 = x_v, \quad h_v^{k+1} = \phi_k(h_v^k, \psi_k(\{h_u^k | u \in \mathcal{N}_v\})), \tag{1}$$

where ψ_k maps the multiset of neighboring node features in a permutation-invariant fashion to a vector, and ϕ_k maps pairs of vectors to a single vector. Popular examples of MPNNs include GIN [15] GAT [16] GCN [17] and many others [18–20].

2 Oversquashing: Long Range and Short Range

To improve the theoretical understanding of oversquashing, we study a family of synthetic graph-transfer problems. We begin by introducing terminology and notation that will be used throughout the examples.

We consider graphs whose node set is a disjoint union $V = S \cup C \cup T$, where S denotes *source nodes*, C denotes *central nodes*, and T denotes *target nodes*. In essence, the goal of these tasks is

to transfer information from source nodes to target nodes, with the central nodes serving solely to conduct that information.

Each node feature is a pair $x_v = (\iota_v, \ell_v)$, where $\iota_v \in \{0, 1, \ldots, n\}$ is a *node identifier*, and $\ell_v \in \{1, \ldots, L\}$ is a *node label*. Source nodes have unique identifiers $\iota_v \in \{1 \ldots, n\}$, and their labels represent information to be transferred. Central nodes are all assigned the identifier $\iota_v = 0$. The identifiers of target nodes specify from which source they should receive information (see fig. 1). The identifiers and labels are encoded as one-hot vectors.

We begin with two well-known problems from the graph learning literature. The first problem, **Tree Neighbors-Match**, was introduced by Alon and Yahav [2]. We consider a binary tree, whose source nodes are its leaves, each assigned a distinct identifier ι_s and a label ℓ_s. The root of the tree is connected to a target node t, which is assigned an identifier ι_t (see fig. 1(a)). The goal is to assign to the target node the label of the source node that has the same node identifier. Namely, the MPNN needs to find the leaf node s for which $\iota_s = \iota_t$, and set the output feature h_t^K of node t to ℓ_s.

Surprisingly, Alon and Yahav [2] demonstrated empirically that, as the depth of the tree increases, standard MPNNs struggle to solve this seemingly simple task. They attribute this to the exponential growth in the number of leaves with the depth. Since message passing aggregates information locally, solving the task perfectly requires the root node to encode the information from all leaves, which in turn demands a vector of very high dimension—rendering the approach impractical.

The second problem, **Ring Transfer**, is a simple graph-transfer task introduced by Bodnar et al. [21] and further studied by Di Giovanni et al. [9]. Here, a source node s and a target node t are connected by two paths of length r (see fig. 1(b)). The goal is to transfer the label ℓ_s from the source to the target. Di Giovanni et al. [9] showed that this task also poses difficulties for MPNNs. Their analysis focuses on vanishing gradients rather than on the bottleneck phenomenon. Our first theoretical result, stated below, confirms this intuition: the Ring Transfer task indeed requires long-range interaction and is therefore prone to vanishing gradients. However, it does *not* suffer from the bottleneck effect, in the sense that it does not require high-dimensional node features.

Theorem 1. *For any $r \geq 1$, the Ring Transfer task with radius r requires at least r iterations of an MPNN. However, there exists an MPNN that solves the task exactly whose node feature dimension is independent of r. This also holds if the ring topology is replaced with any other graph.*

proof idea. The necessity of at least r iterations is intuitive and well known. Intuitively, a constant feature dimension is sufficient because all that is needed is to recursively transfer the input source feature vector to neighboring nodes until the target node is reached. For a formal proof, see appendix A. □

2.1 The Two-Radius Problem

We now introduce a new synthetic graph-transfer task, the **Two-Radius problem**. We show that, although it is solvable in theory with only two MPNN iterations, it nevertheless suffers from the bottleneck phenomenon. We further demonstrate that MPNNs struggle to solve it in practice.

We first consider a simple variant of the problem—a family of graphs denoted by $\mathcal{G}_n$, with $n \geq 1$ (see fig. 1(c)). Each graph $\mathcal{G}_n = (V, E, \mathbf{X})$ has a vertex set $V = S \cup C \cup T$ consisting of n source nodes, n target nodes, and a single central node. The source nodes $s \in S$ are assigned distinct identifiers $\iota_s \in \{1, \ldots, n\}$, and the n target nodes $t \in T$ are assigned the same set of identifiers. Each source node is also assigned a distinct label $\ell_s \in \{1, \ldots, n\}$, not necessarily identical to its identifier. The goal is to construct an MPNN such that after K iterations, the output features h_t^K of the target nodes satisfy

$$h_t^K = \ell_s \quad \text{whenever } \iota_t = \iota_s \,.$$

As we show below, this problem can be solved exactly by an MPNN, but only at the cost of a very high feature dimension, of order $n \log n$. This is perhaps unsurprising, since the graphs under consideration are nearly disconnected: removing the single central node disconnects the graph and renders the task impossible to solve by an MPNN. Nonetheless, in the next section, we show that many of the measures proposed in the MPNN literature to assess connectivity and predict oversquashing fail to identify this graph as problematic. Moreover, the graphs in $\mathcal{G}_n$ can be made much better-connected *without* resolving the bottleneck issue.

To show this, we consider a more general family of graphs $\mathcal{G}_{n,k}$ with k central nodes (see fig. 1(d)). Each central node is connected to all source and target nodes. Due to the permutation invariance of MPNNs, the central nodes are indistinguishable. As a result, adding more central nodes does not resolve the bottleneck phenomenon, even though it substantially improves the connectivity of the graph. We formalize this in the following theorem.

Theorem 2. *There exists an MPNN with $T = 2$ iterations that exactly solves the transfer task on $\mathcal{G}_{n,k}$. However, when using b-bit floating-point arithmetic, any MPNN that solves the transfer task on $\mathcal{G}_{n,k}$ with T iterations and intermediate node features of dimension d_t must satisfy*

$$\sum_{t=1}^{T} d_t \geq \frac{n}{2b} \log_2(n/2)$$

for every central node $c \in C$.

Proof. Fix some ordering of the source and target node identifiers, and some initial labeling, leading to some annotated graph $G \in \mathcal{G}_{n,k}$. Next, for any permutation $\tau \in S_n$, consider the new problem instance obtained by permuting the labels by τ while leaving the source and target nodes fixed, giving a new graph G_τ. Let $x_c^t(\tau)$ denote the node features in c after t MPNN iterations applied to G_τ, and denote by $\mathbf{t}(\tau)$ the vector of all target nodes $\mathbf{v} = (x_v, v \in T)$ obtained after T MPNN iterations applied to G_τ. Then $\mathbf{v}(\tau)$ is a function of the central nodes

$$\mathbf{v}(\tau) = \mathbf{v}(x_c^t(\tau), t = 1, \ldots, T)$$

where c is any fixed central node (here we use the fact that the nodes at all central nodes are the same, so we can look only at one of them). Since the MPNN solves the task exactly, we know in particular that $\mathbf{v}(\tau) \neq \mathbf{v}(\sigma)$ for any two distinct permutations $\tau \neq \sigma$. Therefore $\mathbf{v}$ can assume $n!$ different values as the permutation τ changes, and therefore, the vector $(x_c^t(\tau), t = 1, \ldots, T)$ can also attain $n!$ different values. For this to be possible, this vector must of dimension sufficient to contain so many values. Namely the total dimension $d = \sum_{t=1}^{T} d_t$ of the vector must satisfy $(2^b)^d \geq n!$, which implies that

$$b \cdot d \geq \log_2(n!) \geq \log_2 \left(\left(\frac{n}{2} \right)^{n/2} \right) = \frac{n}{2} \log_2(n/2)$$

□

As a concluding remark, we note that the high feature dimension required by theorem 2 is not the only difficulty in the problem. As our experiments below show, MPNNs indeed struggle to solve this problem even with high feature dimension. We believe that the main difficulty is the challenge of reliably mapping an input multiset of increasingly large size n of node features into the intermediate representation carried by the central nodes, without incurring significant distortion. The dependence of the distortion of a multiset map on the problem size is discussed, e.g., in [22, Theorem 3.3].

2.2 Empirical Performance of MPNNs on the Two-Radius Problem

We next evaluate empirically whether the bottleneck phenomenon in the Two-Radius problem indeed leads to practical performance degradation. Specifically, we consider the case $k = 1$ with $n \in \{10, 50, 150, 200\}$, using standard MPNN architectures: GCN [17], GIN [15], GAT [16], and GraphSAGE [18]. We evaluate all methods with feature dimensions of 256 and 1024 and three different learning rates, and report, for each method, the best result obtained across these runs. In addition, we evaluate a simple Set Transformer [23] without augmenting with structural/positional encoding. The Set Transformer treats the vertex features as a multiset, while ignoring the edge structure of the graph. Since Transformer allows pairwise interactions between all nodes, it is not expected to suffer from oversquashing. We further include a standard MLP, which, unlike the other methods, is not permutation invariant, and is thus a priori not expected to suffer from oversquashing, although it may lack sufficient inductive bias to generalize well on this task.

The experiment results appear in fig. 2. As seen in the figure, MLPs perform poorly on this task even for small n. MPNN performance degrades as n increases, in line with our analysis. The only methods that succeed are the Transformer, which consistently achieves 100% accuracy on all instances, and GAT, which, with a high embedding dimension, can reach 90% accuracy even when $n = 200$. In

addition to its lower accuracy compared to the Transformer, GAT was substantially more difficult to train: these results required using a very low learning rate and training for many epochs. As shown in fig. 3(a), achieving at least 92% accuracy required more than 100 epochs with GAT, whereas the Transformer needed fewer than 10. A more fine-grained evaluation of the effect of hidden-feature dimension appears in fig. 6 in appendix B.

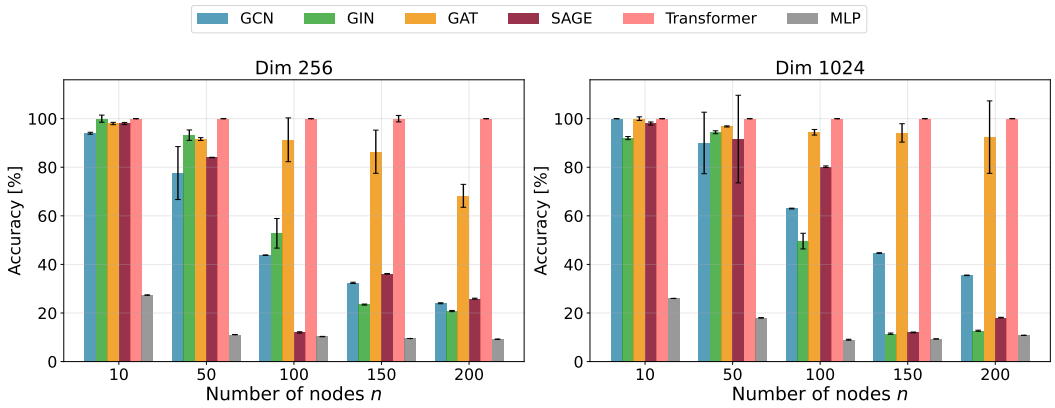
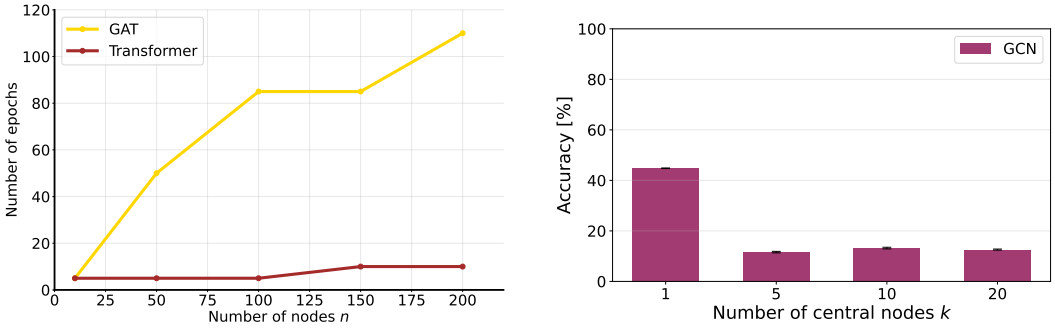


Figure 2: Test accuracy comparison across different models on the Two-Radius problem. Performance is evaluated for varying numbers of nodes $n \in \{10, 50, 150, 200\}$ with hidden dimensions of 256 and 1024. Transformer consistently achieves 100% accuracy while MPNN performance degrades as n increases. Error bars indicate the standard error of the mean.

Next, we examine the effect of changing the number of central nodes k on the performance of MPNNs. As noted earlier, increasing k improves graph connectivity, but it is not expected to improve MPNN performance, because permutation equivariance implies that all central node features are identical across the message-passing process. Empirically, running GCN on the Two-Radius problem with varying k, we found that performance indeed does not improve as k grows and, perhaps surprisingly, is even worse than with $k = 1$. This is shown in fig. 3(b).



(a) (b)

Figure 3: (a) Training efficiency comparison between GAT and Transformer on the Two-Radius problem, measured in the number of epochs required to achieve 92% accuracy. (b) Effect of the number of central nodes k on GCN performance for the Two-Radius problem with $n = 100$. Accuracy remains poor regardless of k, demonstrating that increasing graph connectivity via additional central nodes does not resolve the bottleneck phenomenon. Error bars in (b) (barely visible due to low variance) indicate the standard error of the mean.

In summary, we find that MPNNs struggle to solve the Two-Radius problem, whereas Set Transformer succeeds easily. This strengthens the case for Graph Transformers over MPNNs, and serves as an example of a scenario where disregarding the graph structure can be advantageous—a phenomenon discussed in [24].

3 Oversquashing Measures Don't Explain the Two-Radius Problem

As seen in the previous section, the Two-Radius problem suffers from the bottleneck effect (theorem 2) and MPNNs struggle to solve it in practice. The goal of this section is to revisit common proposals

for measuring and characterizing oversquashing, and to show that most of these measures fail to capture the oversquashing that arises in the Two-Radius problem.

3.1 Current Explanations for Oversquashing

Problem Radius. In the original oversquashing paper [2], the authors argued that the root cause of oversquashing is a large *problem radius*. Their reasoning connects several factors: the literal excessive compression of information in node features (oversquashing), the number of message-passing iterations, and, consequently, both the problem radius and the growth rate of the receptive field of message-recipient nodes, which are, respectively, causes and consequences of the number of MPNN iterations.

In the Two-Radius problem introduced in the previous subsection, the problem radius is fixed at 2, independent of n, and after only two message-passing iterations, the receptive field of all nodes is constant. This setting allows us to *decouple* the effect of large problem radius, which leads to vanishing gradients, from the bottleneck effect, which arises from large receptive fields. We regard these as two distinct mechanisms underlying oversquashing.

Vanishing Gradients. A common explanation of oversquashing is that long range leads to vanishing gradients. In fact, many explanations of oversquashing provide analysis showing how the size of the gradients depends on certain topological properties of the graph, and using this analysis to devise rewiring techniques aimed at improving these topological properties and reducing oversquashing.

The presence of vanishing gradients in the Two-Radius and Ring Transfer problem was evaluated by computing the average gradient norm of the model output (GCN) over 10 randomly selected test samples. The results are shown in fig. 4. For the Ring Transfer problem, both accuracy and gradient norm decrease drastically with the problem radius, whereas for the Two-Radius problem, as n increases, the accuracy of MPNN decreases, but its gradient norm exhibits only minor decrease. This suggests that the vanishing gradient effect is related to long-range interactions, but not to the bottleneck effect. To ensure that the hindered performance in the Two-Radius problem is not caused by oversmoothing, we computed the relative Mean Absolute Difference (MAD) energy [25] over the target nodes (see appendix C for details). As seen in the figure, this energy does not decay toward zero, thereby ruling out oversmoothing as the underlying cause.

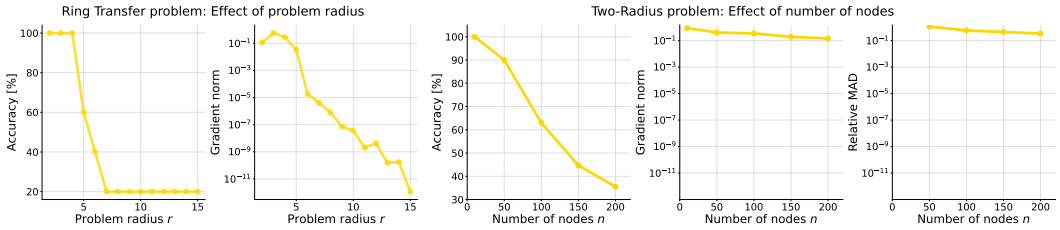


Figure 4: GCN's accuracy deteriorates as the problem radius increases, which is correlated with vanishing gradients. On the other hand, problems with bottlenecks are also difficult for MPNNs, but they do not suffer from vanishing gradients or oversquashing.

Spectral methods. Several works have proposed spectral quantities of the graph Laplacian as indicators of oversquashing, in particular the *spectral gap* (the difference between the first and second eigenvalues of the normalized Laplacian) and the closely related *Cheeger constant*, which measures the presence of sparse cuts in the graph [8, 26, 27]. Intuitively, graphs with a small spectral gap or low Cheeger constant contain bottlenecks that may hinder information flow, and such properties have been argued to correlate with oversquashing. However, in the Two-Radius problem with a single central node, the Cheeger constant is equal to 1 for all values of n. Moreover, when $k \geq 1$ central nodes are taken, the graph becomes more connected, and the Cheeger constant grows with k:

Theorem 3. *The Cheeger constant h_G for the graph $\mathcal{G}_{n,k}$ with $k \leq 2n$ central nodes is lower-bounded by $h_G \geq \frac{k}{8}$.*

Nevertheless, the empirical and theoretical difficulty of the problem clearly increases with n, as shown in fig. 2 and theorem 2, showing that these spectral criteria fail to capture the bottleneck effect responsible for oversquashing in this setting.

Ricci Curvature. The concept of Ricci curvature as an explanation for oversquashing was introduced in Topping et al. [8]. In that work, the authors proved upper bounds on certain gradient norms in terms of this curvature, thereby showing that edges with negative curvature act as "information bottlenecks," which can lead to vanishing gradients.

Specifically, their main relevant result (Theorem 4) states that for any pair of vertices i, j connected by an edge, and for any $\delta > 0$, under some additional assumptions, there exists a nonempty set Q_j of vertices in the two-hop neighborhood of i such that

$$\frac{1}{|Q_j|} \sum_{k \in Q_j} \left| \frac{\partial h_k^2}{\partial h_i^0} \right| < (\alpha \cdot \beta)^2 \cdot \delta^{\frac{1}{4}},$$

where α and β are upper Lipschitz bounds on the update and aggregation functions in the message-passing procedure, and ℓ_0 is any integer between 0 and $L - 2$, with L denoting the number of message-passing iterations. Their result requires

$$\mathrm{Ric}(i, j) + 2 \leq \delta < \max\{d_i, d_j\}^{-\frac{1}{2}},$$

where $\mathrm{Ric}(i, j)$ is the *Ricci* curvature of edge (i, j), also referred to as the *balanced Forman curvature*.

In the Two-Radius problem, however, every edge (i, j) connects a vertex of degree 1 to a vertex of degree n. This implies $\mathrm{Ric}(i, j) = 0$ (see Definition 1 in [8]), and the above assumption becomes

$$2 \leq \delta < n^{-\frac{1}{2}} \leq 1,$$

which is impossible. Hence, their result is not applicable to this problem and, in particular, does not predict oversquashing.

Effective Resistance. A closely related quantity, proposed by Black et al. [10], is the *effective resistance*. Inspired by the concept of resistance in electrical networks, the effective resistance between two nodes u and v decreases as the number of paths between them increases, and increases with the lengths of those paths. Formally it is defined as

$$R_{u,v} = (1_u - 1_v)\, L^+ \,(1_u - 1_v)\,, \tag{2}$$

where L^+ is the pseudo-inverse of the non-normalized graph Laplacian. In the special case where u and v are connected by vertex-disjoint paths, the effective resistance can be expressed by a simpler formula (see Figure 1 therein)

$$R_{u,v} = \left(\sum_{p \text{ is a path from } u \text{ to } v} \mathrm{Length}(p)^{-1} \right)^{-1}. \tag{3}$$

To show that high effective resistance leads to oversquashing bottlenecks, the authors proved an upper bound on the Frobenius norm of the Jacobian in terms of the effective resistance between the nodes.

However, in our example, it can be seen from eq. (3) that the effective resistance between the source and destination vertices always equals 2, independently of n, whereas the core phenomenon of oversquashing is clearly aggravated as n increases.

Direct gradient-decay analysis. Di Giovanni et al. [9] further developed the ideas of Black et al. [10] and derived upper bounds on the Jacobian norm directly from basic topological features of the input graph, that is, without passing through spectral graph theory. Their first result, Theorem 3.2, is of the form

$$\| \frac{\partial h_v^{(m)}}{\partial h_u^{(0)}} \| \leq C^m \,(S^m)_{u,v}\,, \tag{4}$$

where C is a constant depending on the model and hidden feature dimension, and S is a matrix derived from the model parameters and the graph adjacency matrix. When the right-hand side of eq. (4) decays exponentially with m, their theorem predicts exponential decay of the Jacobian norm. However, in our setting, the number of message-passing iterations is constant $m = 2$, so the theorem does not predict exponential gradient decay.

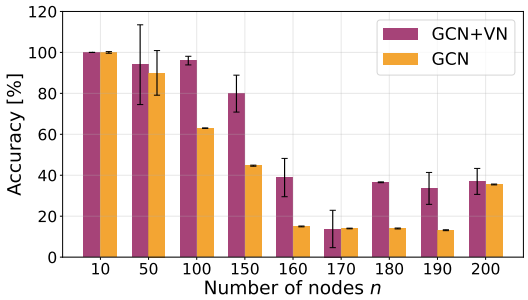


Figure 5: Comparison of GCN performance with and without virtual nodes (VN) on the Two-Radius problem. While virtual nodes provide modest improvements, performance still degrades significantly as n increases, indicating that VNs do not fully address the bottleneck in short-range oversquashing. Error bars indicate the standard error of the mean.

In another result [9, Theorem 4.1], they present a bound that similarly predicts exponential gradient decay, with the exponent depending on the distance between the two nodes and the number of message-passing iterations. Since both quantities in our example are constant, this theorem does not predict the observed oversquashing.

The remaining results in [9] similarly assume either large distances or large number of message-passing iterations, both of which do not hold in our example.

In fig. 1(b), it can be seen that for the Ring Transfer problem, the vanishing-gradient bounds do capture the oversquashing effect demonstrated empirically in fig. 4.

4 Virtual Nodes and Transformers

Several papers have recently argued, both theoretically [7, 12, 28, 29] and empirically [28, 30, 31], that MPNNs with virtual nodes (VNs) can effectively handle oversquashing. In the context of long-range oversquashing, this is indeed reasonable, since virtual nodes reduce the effective problem radius to 2. However, for the Two-Radius problem, a virtual node would act similarly to the existing central nodes in these graphs, so adding a virtual node is not expected to yield substantial improvement for MPNNs.

To evaluate this hypothesis, we compared GCN with and without a virtual node, as shown in fig. 5. As seen in the figure, while adding a virtual node does seem to improve performance for some values of n, GCN+VN still fails to solve the Two-Radius problem for larger values of n. Thus, the Two-Radius problem provides a synthetic example in which MPNN+VN is less effective than Transformers.

5 Conclusion

In this work, we revisited the phenomenon of oversquashing in MPNNs and demonstrated that it is not restricted to long-range tasks. Through the Two-Radius problem, we identified a setting where oversquashing arises even in short-range scenarios and showed that this corresponds to a bottleneck effect, separate from the vanishing gradient effect that dominates in long-range tasks.

Our theoretical analysis established that solving the Two-Radius problem with MPNNs requires large feature dimensions that depend on the graph size, and our empirical results confirmed that standard MPNNs, even when augmented with virtual nodes, struggle on this task. By contrast, Graph Transformers and related architectures succeed, underscoring their potential as a more robust solution to oversquashing in settings where MPNNs remain bottlenecked.

Our results also clarify the limitations of existing measures of connectivity as predictors of over-squashing, and highlight the need for refined metrics that account for short-range bottlenecks. We hope that the Two-Radius problem will serve as a useful benchmark for oversquashing in future studies, and that our analysis can guide the design of new architectures that combine the efficiency of MPNNs with the expressivity of transformers.

References

[1] T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023. 1, 3

[2] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=i80OPhOCVH2. 1, 2, 3, 4, 7

[3] Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In *NeurIPS*, 2021. 1, 2

[4] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In *NeurIPS*, 2022. 2

[5] Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J. Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *ICML*, 2023. 2

[6] Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. In *NeurIPS*, 2022. 1, 2

[7] Chen Cai, Truong Son Hy, Rose Yu, and Yusu Wang. On the connection between MPNN and graph transformer. In *International conference on machine learning*, pages 3408–3430. PMLR, 2023. 2, 9, 13

[8] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *ICLR*, 2022. 2, 7, 8

[9] Francesco Di Giovanni, Lorenzo Giusti, Federico Barbero, Giulia Luise, Pietro Lio, and Michael M Bronstein. On over-squashing in message passing neural networks: The impact of width, depth, and topology. In *ICML*, pages 7865–7885, 2023. 2, 4, 8, 9

[10] Mitchell Black, Zhengchao Wan, Amir Nayyeri, and Yusu Wang. Understanding oversquashing in gnns through the lens of effective resistance. In *ICML*, pages 2528–2547, 2023. 2, 8

[11] Ashok K Chandra, Prabhakar Raghavan, Walter L Ruzzo, and Roman Smolensky. The electrical resistance of a graph captures its commute and cover times. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 574–586, 1989. 2

[12] Eran Rosenbluth, Jan Tönshoff, Martin Ritzert, Berke Kisin, and Martin Grohe. Distinguished in uniform: Self-attention vs. virtual nodes. In *ICLR*, 2024. 2, 9

[13] Jan Tönshoff, Martin Ritzert, Eran Rosenbluth, and Martin Grohe. Where did the gap go? reassessing the long-range graph benchmark. In *Learning on Graphs Conference*, 2023. 2

[14] Adrian Arnaiz-Rodriguez and Federico Errica. Oversmoothing," oversquashing", heterophily, long-range, and more: Demystifying common beliefs in graph machine learning. *arXiv preprint arXiv:2505.15547*, 2025. 3

[15] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *ICLR*, 2019. 3, 5

[16] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *ICLR*, 2018. 3, 5

[17] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*, 2017. 3, 5

[18] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *NIPS*, pages 1024–1034, 2017. 3, 5

[19] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. In *ICLR*, 2016.

[20] Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. Gated Graph Recurrent Neural Networks. In *IEEE Transactions on Signal Processing*, 2020. 3

[21] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Liò, Guido F Montufar, and Michael Bronstein. Weisfeiler and Lehman go cellular: CW networks. In *NeurIPS*, 2021. 4

[22] Yair Davidson and Nadav Dym. On the hölder stability of multiset and graph neural networks. In *The Thirteenth International Conference on Learning Representations*, 2025. 5

[23] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In *ICML*, pages 3744–3753, 2019. 5, 13

[24] Maya Bechler-Speicher, Ido Amos, Ran Gilad-Bachrach, and Amir Globerson. Graph neural networks use graphs when they shouldn't. In *Forty-first International Conference on Machine Learning*, 2024. 6

[25] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, pages 3438–3445, 2020. 7, 13

[26] Pradeep Kr Banerjee, Kedar Karhadkar, Yu Guang Wang, Uri Alon, and Guido Montúfar. Oversquashing in gnns through the lens of information contraction and graph expansion. In *Allerton*, pages 1–8. IEEE, 2022. 7

[27] Kedar Karhadkar, Pradeep Kr Banerjee, and Guido Montúfar. Fosr: First-order spectral rewiring for addressing oversquashing in gnns. *arXiv preprint arXiv:2210.11790*, 2022. 7

[28] Chendi Qian, Andrei Manolache, Christopher Morris, and Mathias Niepert. Probabilistic graph rewiring via virtual nodes. In *NeurIPS*, 2024. 9

[29] Joshua Southern, Johannes Lutzeyer, Fabrizio Frasca, and Michael M Bronstein. Understanding virtual nodes: Oversquashing and node heterogeneity. In *ICLR*, 2025. 9

[30] Trang Pham, Truyen Tran, Hoa Dam, and Svetha Venkatesh. Graph classification via deep learning with virtual nodes. In *Proceedings of the Australasian Joint Conference on Artificial Intelligence*, 2017. 9

[31] EunJeong Hwang, Veronika Thost, Shib Sankar Dasgupta, and Tengfei Ma. An analysis of virtual nodes in graph neural networks for link prediction. In *The First Learning on Graphs Conference*, 2022. 9, 13

A Proofs

Proof of Theorem 1. The setting we are considering is that we are given some graph G, with a single source node s with node features $x_s = (1, \ell_s)$, where 1 is the node identifier, signifying that this is the source node, and ℓ_s is the source label coming from some finite alphabet $\{1, \dots, A\}$. All other nodes v of the graph are given an initial value $x_v = (0, \ell_v)$, where ℓ_v comes from the same alphabet (and is irrelevant to the problem at hand). There is one node, the target node t, whose shortest path distance from the node is r, and the goal is for an MPNN to update the target node to achieve the original value of the source node, namely $h_t^k = x_s$.

It is clear that in less that r iterations, an MPNN will not be able to achieve this goal. However, in r iterations this goal is easily achieved by an MPNN whose feature dimension does not depend on r. Namely, we begin as always with the initial features $h_v^0 = x_v$, and iteratively define $h_v^{k+1} \in \mathbb{R}^2$ by

$$h_v^{k+1}[1] = \max_{u \in \mathcal{N}_v \cup \{v\}} h_u^k[1], \quad h_v^{k+1}[2] = \max_{u \in \mathcal{N}_v \cup \{v\}} h_u^k[1] \cdot h_u^k[2]$$

We can then prove recursively on k that if the distance of v from s is $\leq k$, then $h_v^k = h_s^0$, and otherwise $h_v^k[1] = 0$.

For $k = 0$ this is clearly true. If the claim is true for k, then for $k+1$ we have that, if the distance of v from s is more than $k+1$, than all its neighbors u will all have distance more than k from s, and so $h_u^k[1] = 0$ and therefore $h_v^{k+1}[1] = 0$. If v is a node of distance $\leq k+1$ from s, then there is some $u \in N(v) \cup \{v\}$ whose distance from s is $\leq k$. As a result $h_v^{k+1}[1] = 1 = h_s^0, h_v^{k+1}[2] = h_u^k[2] = h_s^0[2]$.

In particular, it follows that after r iterations the target node will obtain the value of the source node. This concludes the proof. □

Proof of Theorem 3. Recall that for a graph G, and a subset $A \subseteq V$, the boundary ∂A is defined as the number of edges between nodes in A and nodes in the complement of A, and the Cheeger constant is defined as

$$h_G = \min_{A \subseteq V, 0 < |A| \leq |V|/2} \frac{|\partial A|}{|A|}.$$

Now let $A \subseteq V$ with $0 < |A| \leq |V|/2$. Denote the number of central nodes in A by a and the number of source and target nodes by b. By assumption

$$a + b \leq |V|/2 = n + k/2.$$

We now consider three cases: **case 1:** If $a \geq k/2$, then necessarily $b \leq n$. It follows that

$$\frac{|\partial A|}{|A|} \geq \frac{a \cdot (2n - b)}{a + b} \geq \frac{nk/2}{4n} = \frac{k}{8}$$

case 2: If $b \geq n$ then necessarily $a \leq k/2$. It follows that

$$\frac{|\partial A|}{|A|} \geq \frac{b \cdot (k - a)}{a + b} \geq \frac{n \cdot k/2}{4n} = \frac{k}{8}.$$

case 3: If both $b \leq n$ and $a \leq k/2$. Then

$$\frac{|\partial A|}{|A|} = \frac{a(2n - b) + b(k - a)}{a + b} \geq \frac{na + bk/2}{a + b} \geq k/2.$$

This concludes the proof. □

B Additional Experimental Results

To complement the analysis in the main text, here we presents a more fine-grained evaluation of the effect of the hidden-feature dimension on learning accuracy (fig. 6).

The results follow the same experimental setup as fig. 2 and further illustrate that increasing the number of nodes makes the problem challenging for MPNNs, whereas the Transformer remains robust, with GAT ranking between the two. As shown in the figure, this difficulty is partially mitigated by increasing the hidden-feature dimension, which improves the performance of both MPNNs and GAT but does not close the gap with the Transformer. Overall, these results reinforce that MPNNs are limited by the bottleneck effect, whereas the Transformer remains largely unaffected.

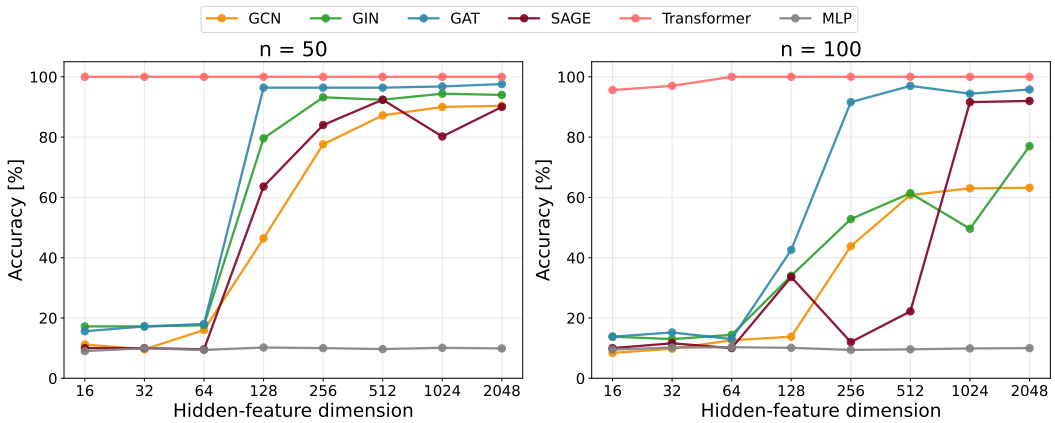


Figure 6: Fine-grained evaluation of the effect of hidden-feature dimension on learning accuracy in the same experimental setting as fig. 2.

C Experimental Details

Input Representation

The input representation consists of two concatenated one-hot vectors: one encoding the ID and another encoding the label. This combined one-hot representation is fed directly into the initial MPNN layer, which maps it from the input dimensionality to the hidden dimension. Our implementation does not use a separate encoder or projection layer for this operation.

Mean Absolute Difference (MAD)

In fig. 4 we included a relative variant of the Mean Absolute Difference (MAD)—an energy proposed by Chen et al. [25] to assess the extent to which oversmoothing takes place. For the output feature vectors h_v^K of the target nodes $v \in T$, it is computed by:

$$\frac{\frac{1}{\frac{1}{2}|T|(|T|-1)} \sum_{u \neq v \in T} \|h_u^K - h_v^K\|}{\frac{1}{|T|} \sum_{v \in T} \|h_v^K\|}, \tag{5}$$

namely, the average pairwise distance between the target node features, divided by the average norm of those features.

Hardware and Software

All experiments were conducted on an NVIDIA A40 GPU with 48GB memory, using CUDA 12.8. We implemented all models using PyTorch with PyTorch Geometric for graph operations and PyTorch Lightning for training management.

Training procedure

All models are trained using Adam optimizer with cross-entropy loss, running for a maximum of 1000 epochs with ReduceLROnPlateau learning rate scheduler.

Model Architectures and Initialization

We evaluated the Two-Radius problem with node configurations $n \in \{10, 50, 100, 150, 200\}$ and central nodes $k = 1$ (default), with additional ablation studies using $k \in \{5, 10, 20\}$. For reproducibility, we fixed the seed to be 0.

Implementation details

- For the Two-Radius problem, we masked source and central nodes during evaluation, computing accuracy only on target nodes.
- In gradient norm computation, we use the first 5 test samples to reduce computational cost.
- **Central nodes (k).** Fixed $n = 100$, varied $k \in \{1, 5, 10, 20\}$ using GCN with lr = 5×10^{-4}.
- **Node Identifiers and Labels.** For the Two-Radius problem, nodes are initialized as follows:
  - All identifiers and labels are encoded as one-hot vectors.
  - *Source nodes:* Each source node $s \in S$ receives a unique identifier ι_s randomly sampled without replacement from $\{1, \ldots, n\}$, and a label ℓ_s, chosen randomly from $\{1, \ldots, L\}$ where $L = 10$ is the number of classes in our experiments.
  - *Target nodes:* Each target node $t \in T$ receives a unique identifier ι_t from $\{1, \ldots, n\}$, chosen randomly without replacement. Labels are assigned a constant value L, chosen arbitrarily.
  - *Central nodes:* In the case of one central node c, the node was assigned the identifier $\iota_c = n + 1$. In the case of k central nodes, identifiers were assigned $\{n + 1, \ldots, n + k\}$. All central nodes were labeled $\ell_c = 0$.
- **Virtual Nodes (VNs).** To test whether VNs mitigate the short-range bottleneck, we augment GCN with virtual nodes following Cai et al. [7] and Hwang et al. [31]. In the single-VN variant, the VN is initialized to zero and, after each layer, a two-layer MLP updates it; the resulting VN embedding is then added (broadcast) to every node representation. In the multiple-VN variant, each VN has its own MLP and is updated independently, and at each layer the sum of all VN embeddings is added to every node. The VN update uses a global aggregation over node features (either mean or sum). We compare GCN with and without VNs using tuned learning rates (5×10^{-4} with VN; 1×10^{-4} without).
- **Set Transformer.** Following Lee et al. [23], we implement a set-based transformer that *ignores edges* and operates solely on node features,treating the input as a multi-set. This Set Transformer architecture, which can be viewed as a Graph Transformer operating without explicit edge

information, processes nodes as an unordered collection where multiplicities matter. In the Two-Radius setting, the model receives one-hot node features (IDs and labels) without connectivity information; thus, source/target/central roles must be inferred from feature patterns. All nodes interact simultaneously via self attention, so any node can attend to all other nodes without routing information through central nodes.

This design is particularly effective here because target nodes can directly attend to sources with matching IDs, bypassing the central-node bottleneck that constrains MPNNs.

Hyperparameter Selection

Table 1 presents the hyperparameter configurations used for all model architectures in our experiments. These parameters were selected through preliminary experiments that balanced memory constraints, computational runtime, and model performance. Each configuration represents the optimal trade-off between these factors for the respective architecture. Table 2 shows the best-performing learning rates for each model across different hidden dimensions (256 and 1024), determined by evaluating three candidate learning rates per configuration and selecting the one yielding the highest validation accuracy. Together, these hyperparameter choices ensure reproducible and fair comparisons across all evaluated models.

Table 1: Model Hyperparameters

Hyperparams	GCN	GAT	GIN	GraphSAGE	MLP	Set Transformer
batch_size	64	32	64	64	64	64
train_samples	7000	7000	7000	7000	7000	7000
test_samples	700	700	700	700	700	700
layers	4	4	4	4	4	2
activation	LeakyReLU	LeakyReLU	LeakyReLU	LeakyReLU	ReLU	ReLU
residual	✓	✓	✓	✓	✓	✓
use_layer_norm	✓	✓	✓	✓	✓	✓
use_activation	✓	✓	✓	✓	✓	✓
Attention_heads	–	2	–	–	–	2
lr_factor	0.1	0.3	0.5	0.1	0.1	0.5
dropout	–	0.1	–	–	0.3	0.1

Table 2: Optimal Learning Rates

Model	Hidden Dimension	
	256	1024
GCN	5×10^{-4}	1×10^{-4}
GAT	1×10^{-4}	5×10^{-5}
GIN	5×10^{-5}	5×10^{-4}
GraphSAGE	5×10^{-5}	1×10^{-4}
MLP	1×10^{-4}	1×10^{-4}
Set Transformer	1×10^{-3}	1×10^{-3}