gLSTM: Mitigating Over-Squashing by Increasing Storage Capacity

Anonymous authors
Paper under double-blind review

Abstract

Graph Neural Networks (GNNs) leverage the graph structure to transmit information between nodes, typically through the message-passing mechanism. While these models have found a wide variety of applications, they are known to suffer from *over-squashing*, where information from a large receptive field of node representations is collapsed into a single fixed sized vector, resulting in an information bottleneck. In this paper, we re-examine the over-squashing phenomenon through the lens of model *storage and retrieval capacity*, which we define as the amount of information that can be stored in a node's representation for later use. We study some of the limitations of existing tasks used to measure over-squashing and introduce a new synthetic task to demonstrate that an information bottleneck can saturate this capacity. Furthermore, we adapt ideas from the sequence modeling literature on associative memories, fast weight programmers, and the xLSTM model to develop a novel GNN architecture with improved capacity. We demonstrate strong performance of this architecture both on our capacity synthetic task, as well as a range of real-world graph benchmarks.

1 Introduction

Graph Neural Networks (GNNs) (Sperduti, 1993; Gori et al., 2005; Scarselli et al., 2008; Micheli, 2009; Bruna et al., 2014; Defferrard et al., 2017) have emerged as a standard framework for learning on graph-structured data. The majority of these models follow a *message passing* paradigm, where nodes iteratively exchange information with neighbors, commonly referred to as Message-Passing Neural Networks (MPNNs). Examples of this family of architectures include GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), GIN (Xu et al., 2018), and GraphSAGE (Hamilton et al., 2017).

Since each MPNN layer exchanges information between neighboring nodes to update node representations, the number of layers thus dictates the receptive field: the set of nodes over which information is aggregated. Deep MPNNs are, in theory, desirable as they can model long-range dependencies, but scaling to many layers has historically been difficult due to two pervasive issues that have received significant attention in the literature: *over-smoothing* and *over-squashing*. We focus on the latter in this work.

Over-squashing was initially identified by Alon & Yahav (2021) as a problem of capacity; the compression of information from a node's receptive field into a fixed-size vector. This was linked with depth and long-range dependencies, since on many graphs receptive fields grow exponentially with depth. Later work (Topping et al., 2022; Di Giovanni et al., 2023a) identified that this bottleneck could also result in low sensitivity as measured by the node Jacobian, linking graph topology and aspects of model architecture via an upper bound on this Jacobian. This low sensitivity arises due to repeated degree normalization and application of a contractive nonlinearity over many layers. We argue (Sections 3 and 5.2) that these two issues – limited information storage capacity, and low sensitivity – are distinct. This work revives the capacity viewpoint of over-squashing which has received little attention since the work of Alon & Yahav (2021), and studies it in *isolation* from sensitivity issues. We believe this focus on capacity not only provides a more complete understanding of over-squashing but also highlights new directions to mitigate it.

To combat over-squashing, existing research has focused on ameliorating topological bottlenecks through *rewiring* (Gasteiger et al., 2019; Gutteridge et al., 2023; Nguyen et al., 2023) and controlling the flow of information (Bresson & Laurent, 2017; Finkelshtein et al., 2024; Errica et al., 2025)

– both designed to reduce bottlenecks in the computational graph. However, this bottleneck is only an issue if it (1) results in low sensitivity or (2) saturates the storage capacity. We target the latter failure mode: adapting the MPNN architecture to improve its *ability to store and retrieve information*. Framing over-squashing as a capacity limitation that can be addressed at the architecture level exposes a previously unexplored path, and our results validate this direction.

To improve MPNN storage capacity we turn to the sequence modeling literature, which has a long history of tackling equivalent problems (Hochreiter & Schmidhuber, 1997; Orvieto et al., 2023; Gu & Dao, 2023; Beck et al., 2024; Arora et al., 2024). Taking inspiration from these works, we introduce an MPNN architecture that utilizes associative memory (Beck et al., 2024; Schlag et al., 2021; Hopfield, 1982), and demonstrate that this exhibits improved storage capacity.

Contributions. Our main contributions are as follows. In Section 3, we **re-characterize over-squashing as a combination of two distinct issues**: *saturating capacity* and *low sensitivity*, which we term capacity over-squashing and sensitivity over-squashing respectively. We discuss in Section 3.1 the pitfalls of widely used over-squashing tasks, which either fail to evaluate capacity at all, or evaluate the two issues in tandem and are thus unable to separate their effects. In Section 3.2, we introduce a novel synthetic task, which to our knowledge is the first that **measures capacity over-squashing in isolation**. In Section 4, we present a new MPNN architecture based on the recent xLSTM architecture (Beck et al., 2024), which uses **associative memory to increase capacity**, explicitly targeting this capacity over-squashing viewpoint. Section 5 demonstrates that this architecture performs well on our synthetic capacity task and a range of real-world benchmarks, and Section 5.2 demonstrates empirically that **capacity over-squashing can occur separately from sensitivity over-squashing**.

2 Background and Related Work

Message Passing Neural Networks Let a graph $\mathcal{G}$ be a tuple $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ the set of edges. An edge from node u to v is denoted $(u, v) \in \mathcal{E}$. The connectivity is encoded by the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, where $\boldsymbol{A}_{uv} = 1$ if $(u, v) \in \mathcal{E}$ and 0 otherwise. Each node v has a feature vector $\boldsymbol{x}_v \in \mathbb{R}^d$.

GNNs are functions $f_{\boldsymbol{\theta}} : (\mathcal{G}, \{\boldsymbol{x}_v\}) \mapsto \boldsymbol{y}$ with parameters $\boldsymbol{\theta}$, trained via gradient descent to predict node- or graph-level labels $\boldsymbol{y}$. These models typically take the form of MPNNs, which compute latent representations by composing L layers of the following "message passing" node-wise operation:

$$\boldsymbol{h}_u^{(l)} = \phi^{(l)}\big(\boldsymbol{h}_u^{(l-1)},\ \psi^{(l)}(\{\boldsymbol{h}_v^{(l-1)} : (u, v) \in \mathcal{E}\})\big), \tag{1}$$

where $\psi^{(l)}$ is a permutation-invariant *aggregator*, $\phi^{(l)}$ combines neighbor messages with the previous embedding and $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$. Throughout, we use "GNN" and "MPNN" interchangeably. Note we depart from the more usual notation of k for layer index to avoid confusion with *keys*, introduced in Section 3.2. The most commonly used aggregation function takes the form

$$\psi^{(l)}(\{\boldsymbol{h}_v^{(l-1)} : (u, v) \in \mathcal{E}\}) = \sum_v \boldsymbol{O}_{uv}\, \boldsymbol{h}_v^{(l-1)}, \tag{2}$$

where $\boldsymbol{O} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is some message-passing matrix. For GCN (Kipf & Welling, 2017), $\boldsymbol{O} = \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$ with $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ for diagonal $\tilde{\boldsymbol{D}} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ with $\tilde{\boldsymbol{D}}_{ii} = \sum_j \tilde{\boldsymbol{A}}_{ij}$. We frequently denote the set of message-passing neighbors of node u as $\mathcal{N}_u = \{v \in \mathcal{V} \mid \boldsymbol{O}_{uv} \neq 0\}$ – if the message-passing matrix is layer-dependent, we may superscript this with a layer index.

Hopfield Networks Hopfield networks (Hopfield, 1982) were introduced as associative memories storing binary patterns via a "Hebbian learning" rule in which patterns are directly encoded via an outer product. Later modifications increased storage capacity (Krotov & Hopfield, 2016; 2018) or adapted to continuous states (Hopfield, 1984; Koiran, 1994). Retrieval is typically a multi-step iterative process; Ramsauer et al. (2021) introduced a variant with the ability to retrieve patterns in a single step, demonstrating equivalence with Transformer (Vaswani et al., 2017) key-value recall.

Fast Weight Programmers Fast Weight Programmers (FWPs) are a class of neural network motivated by the idea of allowing variable network weights dependent on the input - termed *fast weights*. One method to "program" the fast weights is to take outer products of learned projections of the

input (Schmidhuber, 1992). Schlag et al. (2021) observe that – up to normalization and activation function differences – linear Transformers (Katharopoulos et al., 2020) are equivalent to FWPs.

xLSTM: Associative Memory for Language Modeling Recent work Beck et al. (2024; 2025) introduced xLSTM, a development of the original LSTM Hochreiter & Schmidhuber (1997) architecture that resulted in a performant recurrent neural network capable of language modeling. Of relevance to our work are the following LSTM limitations that xLSTM aims to address: the inability to "revise storage decisions" and the limited storage capacity of the scalar cell states. The first of these issues is addressed through modifying the original LSTM gating to use exponential activation functions rather than sigmoid functions. The second is addressed by introducing associative memory, updated using an outer product update rule equivalent to that of FWPs to store keys and values (see Appendix A for more details).

3 The Two Faces of Over-Squashing

Over-squashing was initially introduced by Alon & Yahav (2021) as an issue of **storage capacity**. They observed that recurrent sequence models exhibit a bottleneck in representing all the information from their past inputs, and this bottleneck exists in a more harmful form in GNNs, in which the information receptive field grows exponentially. They introduced a synthetic task to measure over-squashing by propagating information through various sizes of binary tree.

Later research identified that this computational graph bottleneck *also* resulted in **low sensitivity** and issues of signal propagation. Topping et al. (2022); Di Giovanni et al. (2023a) quantified this low sensitivity via the Jacobian of node representations, establishing the following sensitivity bound: for an MPNN with l layers, c_σ Lipschitz constant of the activation, w maximal entry-value over weight matrices, d embedding dimension and $u, v \in \mathcal{V}$, one has

$$\left\| \frac{\partial \boldsymbol{h}_v^{(l)}}{\partial \boldsymbol{h}_u^{(0)}} \right\|_1 \leq \underbrace{(c_\sigma w d)^l}_{\text{model}} \overbrace{\left(\boldsymbol{O}^l\right)_{uv}}^{\text{topology}}, \tag{3}$$

where $\boldsymbol{O}$ is the message passing matrix used by the MPNN as in Equation (2). This bound establishes that low sensitivity results from both graph topology as well as factors intrinsic to the MPNN model. In particular, sensitivity is lowered by the nature of the message-passing, where the culprit is successive powers of a degree-normalized adjacency matrix. It is also lowered by the contractive nature of the nonlinearity σ and the values of the weight matrices, as established in (Arroyo et al., 2025). Despite this analysis being purely one of sensitivity rather than capacity, it was also termed over-squashing, and has been successful in establishing links to other areas, including the expressive power of MPNNs (Di Giovanni et al., 2023b) and graph effective resistance (Black et al., 2023).

We argue that there are two *distinct* problems arising from bottlenecks in MPNNs: *reduced sensitivity* (sensitivity over-squashing) and *saturating storage capacity* (capacity over-squashing). While due to the influential paper of Topping et al. (2022) the sensitivity viewpoint on over-squashing has thus far been the predominant approach in the literature; **in this work, we seek to revisit the storage capacity viewpoint** and investigate how this issue can be avoided. We define storage capacity as the amount of information that can be stored in a node's representation for later use: a representation is *saturated* when it is unable to store any more information.

Conflation With Depth The vast majority of existing research links over-squashing with depth. To an extent, this is justified: the bound of Equation (3) decreases exponentially with MPNN depth, and real-world graphs tend to exhibit receptive fields that grow exponentially in depth, leading to capacity quickly becoming a problem for deep MPNNs. However, alongside recent work (Arnaiz-Rodriguez & Errica, 2025), we highlight that over-squashing is not *exclusively* a problem of depth: bottlenecks can be observed in single-layer GNNs acting on high-degree nodes – we exploit this fact in our synthetic task of Section 3.2. Furthermore, in studying over-squashing only in the *deep regime*, much of the literature has conflated the problem with issues of *vanishing gradients*, themselves closely linked to the related problem of *over-smoothing* Di Giovanni et al. (2023b). Arroyo et al. (2025) give a more precise treatment of how the issue of over-squashing relates to depth, through over-smoothing (zero collapse) and vanishing gradients. In this work we study over-squashing in the *shallow regime*: this allows us to isolate the issue of saturating capacity, avoiding the effects of depth on both reduced sensitivity (Equation (3)) and vanishing gradients.

3.1 Existing Over-Squashing Tasks do not (Only) test Capacity

An instructive way of contrasting sensitivity against capacity is via synthetic tasks. The most common synthetic tasks currently used to assess over-squashing are the RingTransfer tasks of Di Giovanni et al. (2023a). The goal of these tests is for a MPNN to 'transfer' features contained at a target node to a source node, across a large graph distance. Various graphs are tested, in particular a ring of nodes, but the common feature is that there exists a long shortest-path from the source to target node, so the information must be passed through many nodes. All of these exhibit an exponentially growing receptive field of at least 2^k for k layers, since each node is connected to at least two others; repeated aggregation over neighborhoods and application of MPNN layers and nonlinearities makes this a good test of the sensitivity-based view of over-squashing.

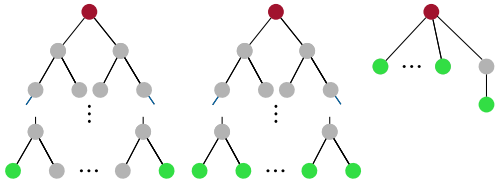


Figure 1: Computational graphs. **Left**: RingTransfer (Di Giovanni et al., 2023a). **Middle**: `Tree-NeighborsMatch` (Alon & Yahav, 2021). **Right**: NAR, introduced in Section 3.2. Nodes with informative features are green, background gray. Output of the red node is trained to solve the task.

However, this task is particularly ill-equipped to test the issue of storage capacity, as the *only* relevant information in the graph is that of the target node, and all intermediate nodes are assigned constant vectors of ones. In this way, there is only a single node's representation worth of information to be transferred. It is unclear how much this task measures behavior found in real-world tasks, as over-squashing will not comprise sensitivity issues in isolation: exponentially growing receptive fields will not be padded by nodes with identical representations containing no new information. Figure 1 (left) visualizes the computational graph of RingTransfer, demonstrating that it is dominated by nodes containing no information.

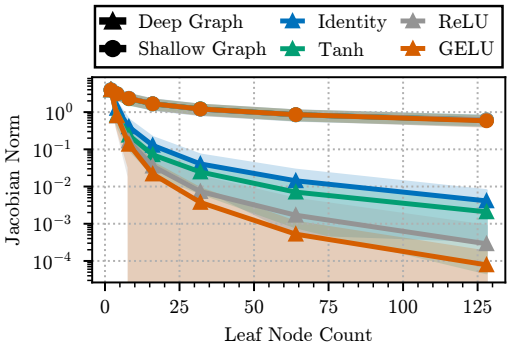


Figure 2: Log Jacobian norms. "Deep" graphs are binary trees of `Tree-NeighborsMatch` (Alon & Yahav, 2021); "Shallow" graphs are single-level trees with the same number of leaves. A GCN of depth equal to the tree depth acts on each. Jacobian norms are $|\partial \boldsymbol{h}_r^{(L)} / \partial \boldsymbol{h}_l^{(0)}|_1$ for root r and leaf l (red/green in Figure 1). Shaded area is standard deviation.

Alon & Yahav (2021) introduced the `Tree-NeighborsMatch` task to measure capacity via propagating information from the leaf nodes of a variable-size binary tree. It shares similarities with the task we introduce in Section 3.2 in that it controls the amount of information that is forced into a bottleneck of a single node representation. However, it propagates this information through a deep binary tree, requiring variable-depth MPNNs to solve this task. This depth significantly worsens sensitivity: we visualize Jacobian norms of a GCN acting on a deep binary tree vs a single layer tree with matching leaf counts in Figure 2, demonstrating that this sensitivity drops off far faster for deep GCNs. This is unsurprising given the bound of Equation (3), showing that the deep GCN must additionally contend with "model" squashing terms of nonlinearity and weight contraction that scale exponentially with depth. Therefore we expect performance degradation trends to be due to both 1) saturating capacity and 2) low sensitivity; deep tasks such as `Tree-NeighborsMatch` are impacted by both over-squashing issues, rather than isolating the issue of capacity.

3.2 Neighbor Associative Recall: Isolating Storage Capacity

We investigate storage capacity by measuring *associative recall*: this is a common approach taken in the sequence-modeling literature (Ba et al., 2016; Schlag et al., 2021; Arora et al., 2024; Jelassi et al., 2024), in which the question of model storage capacity is also clearly of interest. These synthetic tasks involve presenting the model with a sequence of key value pairs followed by a query that corresponds to one of the presented keys, and the model must return the associated value.

To this end, we introduce a task that we refer to as Neighbor Associative Recall (NAR). Whereas the sequence associative recall tasks measure the ability of a model to recall previous information from a variable-length sequence, our graph adaptation is designed to measure the ability of a GNN to recall information from the previous message passing round over a variable number of neighbors.

The task is designed as follows. For a given neighborhood size N we create a graph of $N+3$ nodes. This graph consists of N "neighbor" nodes, a central node to which they are all connected, an intermediate node connected to the central node, and a "query" node connected only to the intermediate node. An example such graph is visualized in Figure 3.

For a fixed neighborhood size N we define a fixed set of keys and values, $N = |\mathbb{K}| = |\mathbb{V}|$, and a pair of learned vector embedding functions $\boldsymbol{e}_k : \mathbb{K} \to \mathbb{R}^{d_{\text{emb}}}$, $\boldsymbol{e}_v : \mathbb{V} \to \mathbb{R}^{d_{\text{emb}}}$ for embedding dimension d_{emb}. Each of the neighbor nodes n has a different assigned key $k_n \in \mathbb{K}$, and also a value $v_n \in \mathbb{V}$, randomly sampled with replacement. The input feature vector of these nodes is a concatenation of the two learned embeddings $\boldsymbol{x}_n = [\boldsymbol{e}_k(k_n); \boldsymbol{e}_v(v_n)] \in \mathbb{R}^{2d_{\text{emb}}}$. The intermediate node and central node both have zero-valued feature vectors. Associated with the query node q is a randomly sampled key-value node m; the input feature vector for the query node consists of the corresponding key embedding concatenated with a vector of zeroes, $\boldsymbol{x}_q = [\boldsymbol{e}_k(k_m); \boldsymbol{0}] \in \mathbb{R}^{2d_{\text{emb}}}$. The model is trained such that the central node must predict the value v_m associated with the sampled key node. Training is via cross-entropy loss where the target of the central node is a one-hot vector corresponding to a fixed ordering of $\mathbb{V}$. This approach can be viewed as a graph adaptation of the sequence associative recall task of Schlag et al. (2021). Results are presented in Section 5.1. An alternative formulation of this task with a regression target is discussed in Appendix B.6.

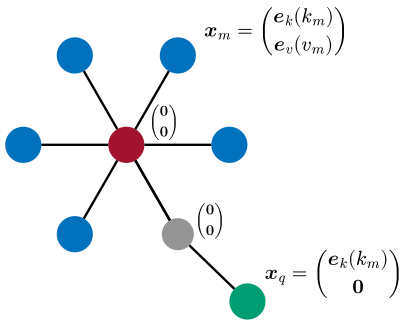


Figure 3: An example graph with $N = 5$ from the NAR task. Key-value nodes are shown in blue, the central node in red and the query node in green. In this graph, m is the randomly sampled index of the key-value node associated with query node q. The target for this graph is a one-hot vector corresponding to v_m.

NAR is designed such that the receptive field of the central node will comprise *only* the neighbor nodes in the first layer. In order to perfectly solve the task, it must store all of the key-value information in this initial receptive field, as it is impossible to limit the scope of the information that might later be required. In the second layer, the receptive field will include the query node: now, the model must selectively recall the correct value from its immediate neighbors.

This task is novel as it assesses over-squashing in the shallow regime: MPNNs tested in Section 5.1 consist of just two message passing layers. This more effectively isolates the issue of capacity, without secondary effects from low sensitivity and vanishing gradients as visualized in Figure 2.

4 gLSTM: Combining Graph Networks and Associative Memory

Prior work on over-squashing has focused almost exclusively on mitigating sensitivity issues, often through graph rewiring (Gasteiger et al., 2019; Gutteridge et al., 2023; Nguyen et al., 2023). By contrast, we find no studies that tackle capacity over-squashing directly. Motivated by memory-capacity gains in sequence models (Ba et al., 2016; Beck et al., 2024), we introduce associative memory into an MPNN architecture to explicitly enlarge its information-storage capacity. We can specifically measure this increased capacity in the graph setting using the NAR task, introduced above. We further introduce the gating scheme of Beck et al. (2024) to investigate its efficacy in the graph setting, given strong sequence modeling performance. Since these adaptations are inspired in part by their successful use in xLSTM, we refer to our related graph architecture as gLSTM.

For any node u at layer l, in addition to the usual MPNN vector hidden state $\boldsymbol{h}_u^{(l)}$, gLSTM maintains a matrix hidden state $\boldsymbol{C}_u^{(l)}$. The initial hidden state $\boldsymbol{h}_u^{(0)}$ is the input node feature vector $\boldsymbol{x}_u$. Keys and values are used to update $\boldsymbol{C}_u^{(l)}$ via an FWP-style outer product rule: these are projections of the previous vector hidden state $\boldsymbol{h}_u^{(l-1)}$. The next vector hidden state $\boldsymbol{h}_u^{(l)}$ is determined by "querying" $\boldsymbol{C}_u^{(l)}$ via matrix multiplication with another projection of the previous vector hidden states.

The modified gLSTM update equations are given below. Highlighted in blue are the differences to xLSTM. Biases correspond exactly to xLSTM (Appendix A) and are omitted for clarity.

State (and normalization) updates:

$$\boldsymbol{C}_u^{(l)} = f_u^{(l)} \boldsymbol{C}_u^{(l-1)} + \sum_{v \in \mathcal{N}_u^{(l)} \cup \{u\}} i_v^{(l)} \boldsymbol{v}_v^{(l)} \otimes \boldsymbol{k}_v^{(l)}$$

$$\boldsymbol{n}_u^{(l)} = f_u^{(l)} \boldsymbol{n}_u^{(l-1)} + \sum_{v \in \mathcal{N}_u^{(l)} \cup \{u\}} i_v^{(l)} \boldsymbol{k}_v^{(l)}$$

$$m_u^{(l)} = \max\left(\left\{\tilde{f}_u^{(l)} + m_u^{(l-1)}\right\} \cup \left\{\tilde{i}_v^{(l)} \mid \forall v \in \mathcal{N}_u^{(l)} \cup \{u\}\right\}\right)$$

Query / Key / Value computation:

$$\boldsymbol{q}_u^{(l)} = \boldsymbol{W}_q \left[\boldsymbol{h}_u^{(l-1)}; \sum_{v \in \mathcal{N}_u^{(l)}} \boldsymbol{h}_v^{(l-1)}\right]$$

$$\boldsymbol{k}_u^{(l)} = \frac{1}{\sqrt{d}} \boldsymbol{W}_k \boldsymbol{h}_u^{(l-1)}$$

$$\boldsymbol{v}_u^{(l)} = \boldsymbol{W}_v \boldsymbol{h}_u^{(l-1)}$$

The square brackets above denote vector concatenation. Concatenating the hidden state for the node and its neighbours in this way keeps them separate and allows the query – which will determine what is retrieved from the matrix memory – to *separately* depend on both the previous state of the node itself and the previous states of its neighbours.

Gate computation:

$$i_u^{(l)} = \exp\left(\tilde{i}_u^{(l)} - m_u^{(l)}\right) \qquad \tilde{i}_u^{(l)} = \boldsymbol{w}_i^T \boldsymbol{h}_u^{(l-1)}$$

$$f_u^{(l)} = \exp\left(\tilde{f}_u^{(l)} + m_u^{(l-1)} - m_u^{(l)}\right) \qquad \tilde{f}_u^{(l)} = \boldsymbol{w}_f^T \boldsymbol{h}_u^{(l-1)}$$

$$\boldsymbol{o}_u^{(l)} = \sigma\left(\tilde{\boldsymbol{o}}_u^{(l)}\right) \qquad \tilde{\boldsymbol{o}}_u^{(l)} = \boldsymbol{W}_o \boldsymbol{h}_u^{(l-1)}$$

Output:

$$\tilde{\boldsymbol{h}}_u^{(l)} = \frac{\boldsymbol{C}_u^{(l)} \boldsymbol{q}_u^{(l)}}{\max\left\{\left|\boldsymbol{n}_u^{(l)\top} \boldsymbol{q}_u^{(l)}\right|, 1\right\}}$$

$$\boldsymbol{h}_u^{(l)} = \boldsymbol{o}_u^{(l)} \odot \tilde{\boldsymbol{h}}_u^{(l)}$$

Block Structure Arroyo et al. (2025) note that sensitivity over-squashing issues are largely caused by vanishing gradients – a phenomenon well-explored in the sequence-modeling literature. In an attempt to address this, gLSTM therefore uses a similar block structure to the mLSTM block upon which it is based. Of particular importance is the residual connection – which brings the norm of the layer-wise Jacobian to the edge of chaos – and use of input and hidden norms, which regulate the magnitude of the Jacobian norms. Figure 4 visualizes the block structure of gLSTM that we employ.

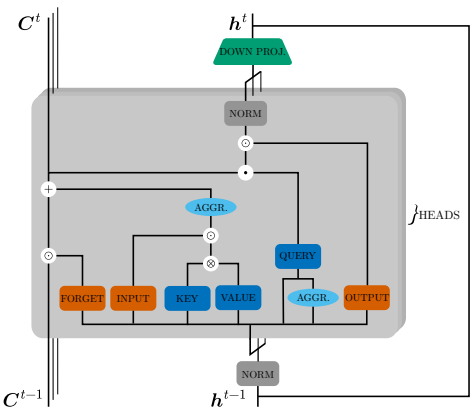


Figure 4: gLSTM block structure. Gate layers shown in orange, query/key/value computation in dark blue. *Aggr.* is short for aggregation, and represents (K-hop) aggregation across neighborhoods. Symbols $\odot, \otimes, +, \cdot$ denote respectively Hadamard product, outer product, vector addition, matrix multiplication.

K-Hop Aggregation Following Arroyo et al. (2025) we combine the memory capabilities of the xLSTM block with a *highly connected* message passing graph structure: employing a k-hop aggregation scheme. In this setting, each node u at layer l will aggregate information from the neighborhood

$$\mathcal{N}_u^{(l)} = \{v \in \mathcal{V} \mid d_{\mathcal{G}}(u, v) = l\},$$

where $d_{\mathcal{G}} : \mathcal{V} \times \mathcal{V} \to \mathbb{R}_{\geq 0}$ is the length of the minimal walk connecting nodes u and v. This approach resembles that of Ding et al. (2024), but with an additional recurrence: hidden states are used as input at each step. This substantially changes the way information can propagate through the graph.

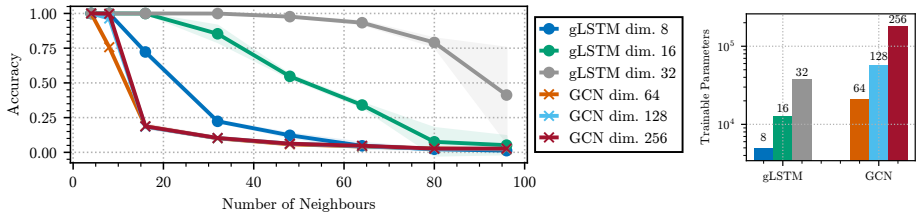
This aggregation scheme appears to greatly improve gLSTM performance: our synthetic task in Section 5.1 significantly benefits from this aggregation scheme, and the ablations in Appendix B.2 demonstrate that it improves performance in all but one of the tested benchmarks. We hypothesize that – in addition to providing a highly connected computational graph that lessens over-squashing sensitivity bottleneck issues – this is because it also provides an extremely useful inductive bias for the *recall* mechanism of gLSTM. Information that has previously been stored in the associative memory is not then included in later message passing rounds, and later nodes are able to query this memory in isolation.

5 EXPERIMENTS

5.1 NEIGHBOR ASSOCIATIVE RECALL

We train various models on NAR with varying neighbor count N, with results shown in Figure 5a. Throughout this section we compare gLSTM using K-hop aggregation to GCN using standard aggregation, since gLSTM performs significantly better in this task when using K-hop aggregation whereas GCN performance is harmed by K-hop. We present additional results in Appendix B.5 where we separate by aggregation method and include results for a larger number of models. A comparison of the number of trainable parameters is shown in Figure 5b. Fair comparison between matrix and vector memory is nontrivial, so we select these parameter counts to "favor" GCN.

These results demonstrate that gLSTM shows significantly improved recall abilities compared to GCN. gLSTM retains perfect recall until the number of neighbors equals the memory dimension of the model: beyond this is where capacity over-squashing appears to become a problem. This agrees with intuition, since the maximum number of orthogonal key vectors (and separately, value vectors) is equal to the memory dimension. However, it is interesting to note how the performance decreases slowly as the neighbor count exceeds this limit, particularly for higher memory dimensions: this appears to be a graph analog of the "graceful saturation" described by Smolensky (1990). By contrast, capacity over-squashing starts much earlier at just $N = 8$ for the largest GCN model tested.



(a) Accuracy

(b) Parameter counts

Figure 5: Test-set mean Accuracy (standard deviation shaded) for the NAR task, for gLSTM and GCN models with various hidden dimensions shown in Figure 5a, number of trainable parameters in Figure 5b. Note that gLSTM uses K-hop aggregation here, whereas GCN does not; see Appendix B.5 for separated performance by aggregation strategy.

5.2 HOW DOES CAPACITY RELATE TO SENSITIVITY?

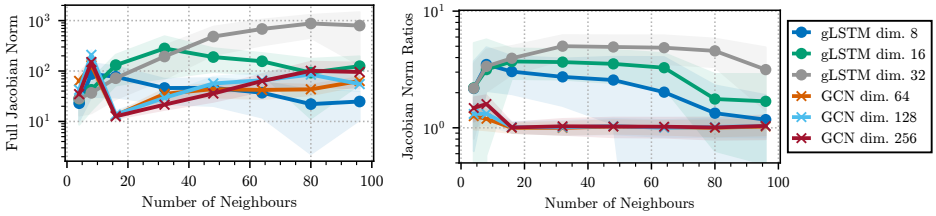
In this section, we investigate empirically how capacity over-squashing – as measured by performance on NAR – relates to sensitivity over-squashing.

We directly measure the Jacobian norm of Topping et al. (2022); Di Giovanni et al. (2023a), computing the sensitivity of the output feature vector on the central (output) node c to the input vectors on the key-value neighbor nodes n, $|\partial \boldsymbol{h}_c^{(2)} / \partial \boldsymbol{x}_n|_1$. These results are visualized in Figure 6a.

We see therefore that sensitivity, as measured by the Jacobian norm, does not correlate with NAR performance. Given that NAR performance degradation is due to capacity over-squashing, we therefore observe that **capacity over-squashing can occur without sensitivity over-squashing**. This is clear from the fact that 1) sensitivity increases consistently for GCN models above $N = 16$ to the point where it matches initial sensitivity, despite no increase in performance and 2) sensitivity for gLSTM tends to carry on increasing beyond where performance starts to degrade. We note these trends – as with all observations we make in this section – hold true for the NAR regression task in Appendix B.6.1.

However, if we examine the difference in Jacobian norms between the neighbor nodes which are *selected* (those which have a key corresponding to the query node) vs *background*, we see trends that align with our notion of capacity. Figure 6a visualizes the ratio of Jacobian norms for selected nodes to that for background nodes. We observe that for all GCN models this ratio quickly falls to unity at the point where capacity over-squashing starts to occur, and gLSTM ratios consistently plateau – and start to slowly decrease – at their memory dimension, similarly coinciding with capacity over-

squashing. It appears therefore that capacity over-squashing harms a model's ability to be selectively sensitive to different nodes in the NAR task.



(a) Jacobian norms (b) Jacobian norm ratios

Figure 6: **Left**: Mean Jacobian norms for different gLSTM and GCN models, with varying number of neighbors in the NAR task. **Right**: Mean ratio between the Jacobian norms of the selected (key corresponds to query) to background (key is different from query) neighbor nodes, for varying model dimensions. Standard deviation shaded in both plots.

Another over-squashing sensitivity metric is that of Di Giovanni et al. (2023b), who introduce the *maximal mixing* metric. For node-level function $\boldsymbol{Y} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$, the mixing of features associated with nodes u, v at a given node i is defined as

$$\underset{\boldsymbol{Y}}{\text{mix}}(i, v, u) = \max_{\boldsymbol{X}} \left\| \frac{\partial^2 \left(\boldsymbol{Y}\left(\boldsymbol{X}\right)\right)_i}{\partial \boldsymbol{x}_u \partial \boldsymbol{x}_v} \right\|.$$

Although motivated through intuition of mixing, we observe the mixed partial derivative can equally be viewed as a composition of partial derivatives quantifying *selective sensitivity* - how much the sensitivity with respect to one node feature varies with respect to another node feature. In this respect, we expect it to be highly relevant to the sensitivity ratios visible in Figures 6b and 9.

To study this empirically for NAR, we take the maximum over the measured Hessians for different models. These Hessian 3-tensors are large, so we further limit to a subset of the overall tensor in order to compute them on available hardware: we are most interested in how the output sensitivity to the neighbor *value* vectors varies with the *query* vector, so we limit to the corresponding input dimensions. For the central, neighbor and query nodes c, n, q this adapted mixing metric is

$$\text{mix}(c, n, q) = \max_{\substack{0\le\alpha<N,\\ 0\le\beta<d_{\text{emb}},\\ d_{\text{emb}}\le\gamma<2d_{\text{emb}}}} \left| \frac{\partial^2 \left(\boldsymbol{h}_c^{(2)}\right)_\alpha}{\partial \left(\boldsymbol{x}_q\right)_\beta \partial \left(\boldsymbol{x}_n\right)_\gamma} \right|,$$

which we plot in Figure 7. We see that gLSTM consistently exhibits greater maximum Hessian values than GCN, and that this collapses for GCN models above 8 neighbors, consistent with the drop in performance. As with the Jacobian ratios, we see plateauing and slow decrease of maximum Hessian values above the memory dimension, but these trends are less pronounced.

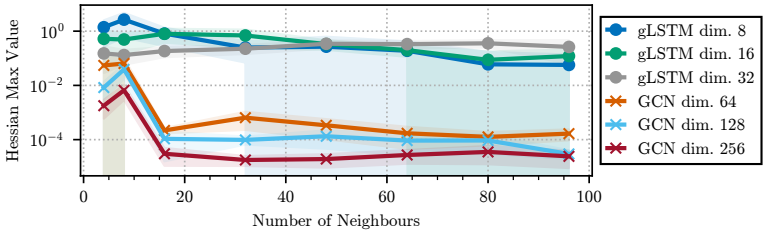


Figure 7: Mean of the maximum Hessian values for different gLSTM and GCN models, averaged across test set examples and different neighbor nodes. Standard deviation shaded.

5.3 LONG RANGE BENCHMARKS

Table 1: Mean and standard deviation of $\log_{10}(\text{MSE})$, averaged over 4 random weight initializations on the GPP tasks from Gravina et al. (2023), from which we report baselines. See Appendix B.1 for discussion of baseline choice. **Top** score in bold, second underlined. Lower is better.

Method	Diam.	Ecc.	SSSP
GCN	0.742 ± 0.047	0.846 ± 0.003	0.950 ± 0.000
GAT	0.822 ± 0.075	0.791 ± 0.022	0.695 ± 0.150
GraphSAGE	0.865 ± 0.40	0.286 ± 0.184	0.786 ± 0.021
GIN	0.613 ± 0.099	0.950 ± 0.001	-0.541 ± 0.419
GCNII	0.529 ± 0.057	0.764 ± 0.036	-1.132 ± 0.013
DGC	0.603 ± 0.005	0.826 ± 0.003	-0.148 ± 0.023
GRAND	0.672 ± 0.049	0.660 ± 0.139	-0.094 ± 0.340
A-DGN	-0.546 ± 0.033	0.305 ± 0.118	**-3.402 ± 0.137**
gLSTM (ours)	**-0.715 ± 0.030**	**-4.036 ± 0.311**	-2.836 ± 0.178
- K-hop	0.042 ± 0.123	0.673 ± 0.021	-3.377 ± 0.142

We evaluate gLSTM on the Graph Property Prediction (GPP) tasks from Gravina et al. (2023) and the Long Range Graph Benchmark (LRGB) from Dwivedi et al. (2022). These benchmarks are both designed to require long range interactions to solve, and thus are an interesting test of the ability of gLSTM to overcome over-squashing and over-smoothing in real world tasks in order to facilitate long range interactions. Performance is reported in Table 1 and Table 2 respectively.

Table 2: Mean and standard deviation on LRGB (Dwivedi et al., 2022), averaged over four random weight initializations. Baselines from the LRGB reevaluation of Tönshoff et al. (2024) and K-hop methods from Arroyo et al. (2025). All methods adhere to a 500k parameter limit. **Top** score in bold, second underlined.

Method	Peptides-Func	Peptides-Struct
	AP (↑)	MAE (↓)
GCN	0.6860±0.0050	**0.2460±0.0007**
GatedGCN	0.6765±0.0047	0.2477±0.0009
GINE	0.6621±0.0067	0.2473±0.0017
GPS	0.6534±0.0091	0.2509±0.0014
K-hop methods		
kGCN-SSM	0.6902±0.0022	0.2581±0.0003
DRew-GCN	0.6804±0.0144	0.2766±0.0019
gLSTM (ours)	**0.7250±0.0023**	0.2527±0.0015

gLSTM achieves comfortably state of the art results on the Diameter and Eccentricity GPP tasks, and very strong performance on SSSP; notably SSSP is the only tested task in which k-hop decreases performance. LRGB results show that gLSTM achieves strong performance in Peptides-Func but relatively weak performance on Peptides-Struct. We hypothesize that the weaker performance on Peptides-Struct may be due to long-range interactions being less relevant for this task, which is very effectively solved by a few-layer GCN. See Appendix B.2 for gLSTM ablations on these benchmarks and Appendix B.3 for details around hyperparameters used.

6 CONCLUSION

In this work, we revisit over-squashing, disambiguating two bottleneck-related issues of sensitivity over-squashing and capacity over-squashing. We introduce a synthetic task that measures capacity over-squashing in isolation and we show that associative memory can improve MPNN capacity. The resulting architecture achieves strong results on real-world benchmarks.

Many avenues for future work remain open. Whereas the sensitivity issue of over-squashing has a mathematical basis via the node Jacobian, to our knowledge the capacity issue does not. Theoretically quantifying this capacity could afford similar directions to those explored via sensitivity, establishing provable links to topology and model properties. With regards to the gLSTM architecture, we translate to a graph setting the gating and associative memory of xLSTM but do not retain the efficiency and parallel training, leaving open future avenues for exploring more efficient MPNN architectures.

Reproducibility Statement

We make available all of our code and experiment configurations to aid reproduction of results. Our experiments utilize the widely-used PyTorch Geometric GraphGym (You et al., 2020) framework which defines a standard framework for MPNN research.

For easiest reproduction of our results, please consult the `readme` in the code repository provided in Appendix B. The repository includes all necessary information to run the experiments: in particular, configs containing the hyperparameters used (also reported in Appendix B.3) and code for all plots used in the paper.

References

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021.

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.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. In *Proceedings of 12th International Conference on Learning Representations (ICLR)*. ICLR, 2024.

Álvaro Arroyo, Alessio Gravina, Benjamin Gutteridge, Federico Barbero, Claudio Gallicchio, Xiaowen Dong, Michael Bronstein, and Pierre Vandergheynst. On vanishing gradients, over-smoothing, and over-squashing in gnns: Bridging recurrent and graph learning. *arXiv preprint arXiv:2502.10818*, 2025.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *Advances in Neural Information Processing Systems*, 37:107547–107603, 2024.

Maximilian Beck, Korbinian Pöppel, Phillip Lippe, Richard Kurle, Patrick M Blies, Günter Klambauer, Sebastian Böck, and Sepp Hochreiter. xLSTM 7b: A recurrent LLM for fast and efficient inference. In *Forty-second International Conference on Machine Learning*, 2025.

Mitchell Black, Zhengchao Wan, Amir Nayyeri, and Yusu Wang. Understanding oversquashing in gnns through the lens of effective resistance. In *International Conference on Machine Learning*, pp. 2528–2547. PMLR, 2023.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs, 2014.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering, 2017.

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 *International conference on machine learning*, pp. 7865–7885. PMLR, 2023a.

Francesco Di Giovanni, T Konstantin Rusch, Michael Bronstein, Andreea Deac, Marc Lackenby, Siddhartha Mishra, and Petar Veličković. How does over-squashing affect the power of gnns? *Transactions on Machine Learning Research*, 2024, 2023b.

Yuhui Ding, Antonio Orvieto, Bobby He, and Thomas Hofmann. Recurrent distance filtering for graph representation learning. In *International Conference on Machine Learning*, pp. 11002–11015. PMLR, 2024.

Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *Advances in Neural Information Processing Systems*, 35:22326–22340, 2022.

Federico Errica, Henrik Christiansen, Viktor Zaverkin, Takashi Maruyama, Mathias Niepert, and Francesco Alesiani. Adaptive message passing: A general framework to mitigate oversmoothing, oversquashing, and underreaching. In *Forty-second International Conference on Machine Learning*, 2025.

Ben Finkelshtein, Xingyue Huang, Michael Bronstein, and İsmail İlkan Ceylan. Cooperative graph neural networks. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 13633–13659, 2024.

Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *Advances in neural information processing systems*, 32, 2019.

M Gori, G Monfardini, and F Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734. IEEE, 2005.

Alessio Gravina, Davide Bacciu, Claudio Gallicchio, et al. Anti-symmetric dgn: a stable architecture for deep graph networks. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023)*, 2023.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Benjamin Gutteridge, Xiaowen Dong, Michael M Bronstein, and Francesco Di Giovanni. Drew: Dynamically rewired message passing with delay. In *International Conference on Machine Learning*, pp. 12252–12267. PMLR, 2023.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

John J Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10):3088–3092, 1984.

Samy Jelassi, David Brandfonbrener, Sham M Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. In *International Conference on Machine Learning*, pp. 21502–21521. PMLR, 2024.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

Pascal Koiran. Dynamics of discrete time, continuous state hopfield networks. *Neural Computation*, 6(3):459–468, 1994.

Dmitry Krotov and John Hopfield. Dense associative memory is robust to adversarial inputs. *Neural computation*, 30(12):3151–3167, 2018.

Dmitry Krotov and John J Hopfield. Dense associative memory for pattern recognition. *Advances in neural information processing systems*, 29, 2016.

Alessio Micheli. Neural Network for Graphs: A Contextual Constructive Approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

Khang Nguyen, Nong Minh Hieu, Vinh Duc Nguyen, Nhat Ho, Stanley Osher, and Tan Minh Nguyen. Revisiting over-smoothing and over-squashing using ollivier-ricci curvature. In *International Conference on Machine Learning*, pp. 25956–25979. PMLR, 2023.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pp. 26670–26698. PMLR, 2023.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Thomas Adler, David Kreil, Michael K Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need. In *International Conference on Learning Representations*, 2021.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pp. 9355–9366. PMLR, 2021.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216, 1990.

Alessandro Sperduti. Encoding labeled graphs by labeling raam. *Advances in Neural Information Processing Systems*, 6, 1993.

Jan Tönshoff, Martin Ritzert, Eran Rosenbluth, and Martin Grohe. Where did the gap go? reassessing the long-range graph benchmark. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *ArXiv*, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Jiaxuan You, Rex Ying, and Jure Leskovec. Design space for graph neural networks. In *NeurIPS*, 2020.

A xLSTM Update Equations

Beck et al. (2024) initially designed xLSTM as a combination of sLSTM blocks and mLSTM blocks - using scalar memory and matrix (associative) memory respectively. However, their follow up work (Beck et al., 2025) uses only mLSTM blocks, and these form the inspiration for gLSTM. Therefore, we will exclusively introduce the mLSTM block update equations in this section.

The update equations, presented in a similar manner to Section 4, are given below.

State (and normalization) updates:

$$\boldsymbol{C}^t = f^t \boldsymbol{C}^{t-1} + i^t \boldsymbol{v}^t \otimes \boldsymbol{k}^t \tag{4}$$

$$\boldsymbol{n}^t = f^t \boldsymbol{n}^{t-1} + i^t \boldsymbol{k}^t \tag{5}$$

$$m^t = \max\left(\tilde{f}^t + m^{t-1}, \tilde{i}^t\right) \tag{6}$$

Query / Key / Value computation:

$$\boldsymbol{q}^t = \boldsymbol{W}_q \boldsymbol{x}^t + \boldsymbol{b}_q \tag{7}$$

$$\boldsymbol{k}^t = \frac{1}{\sqrt{d}} \boldsymbol{W}_k \boldsymbol{x}^t + \boldsymbol{b}_k \tag{8}$$

$$\boldsymbol{v}^t = \boldsymbol{W}_v \boldsymbol{x}^t + \boldsymbol{b}_v \tag{9}$$

Gate computation:

$$i^t = \exp\left(\tilde{i}^t - m^t\right) \qquad \tilde{i}^t = \boldsymbol{w}_i^T \boldsymbol{x}^t + b_i \tag{10}$$

$$f^t = \exp\left(\tilde{f}^t + m^{t-1} - m^t\right) \qquad \tilde{f}^t = \boldsymbol{w}_f^T \boldsymbol{x}^t + b_f \tag{11}$$

$$\boldsymbol{o}^t = \sigma\left(\tilde{\boldsymbol{o}}^t\right) \qquad \tilde{\boldsymbol{o}}^t = \boldsymbol{W}_o \boldsymbol{x}^t + \boldsymbol{b}_o \tag{12}$$

Output:

$$\tilde{\boldsymbol{h}}^t = \boldsymbol{C}^t \boldsymbol{q}^t / \max\left\{\left|\boldsymbol{n}^{t\,T} \boldsymbol{q}^t\right|, 1\right\} \tag{13}$$

$$\boldsymbol{h}^t = \boldsymbol{o}^t \odot \tilde{\boldsymbol{h}}^t \tag{14}$$

B Additional Experiments

Our code for reproducing all experimental results in the main paper and appendices is publicly available at `https://anonymous.4open.science/r/GNN-xLSTM-9A2D`.

B.1 GPP Baselines

We note that due to a subtle PyTorch issue in the original GPP code implementation, normalization is not applied to the dataset targets. A refactor appears to have unknowingly fixed this issue in later iterations of the code so later experiments are run on a normalized variant of the dataset. Unfortunately, this results in unfair comparison, as results can be substantially different between the two variants of the dataset.

Therefore, we test only on the baselines provided in the original GPP paper (Gravina et al., 2023), as we are confident these use the un-normalized variant of the dataset, and this provides us with the largest number of baselines to test against. We additionally ensure that our method uses the same, un-normalized GPP variant.

B.2 Ablations

To identify what elements of the gLSTM architecture are most important for performance on these benchmarks, we perform ablations on the GPP and LRGB datasets. For LRGB, a task with a parameter limit, we ablate in two different settings: the first is simply removing the ablated component,

for which results are presented in Table 4. The second is to scale the hidden dimension $\boldsymbol{h}$ to keep the parameter count as close as possible to the 500k limit - i.e. when removing gating, this will correspondingly increase the hidden dimension. We include these experiments as they more accurately represent the reality of testing a model variant on a task with a parameter limit; these results are presented in Table 5. These two ablation settings show very similar results. GPP ablations are presented in Table 3.

We see that ablating gating only significantly reduces performance on Peptides-Func - other than this, it either leaves performance the same or in some cases, improves performance (GPP Ecc. in particular).

Table 3: Ablation of gLSTM performance on Diam, Ecc, and SSSP from the GPP benchmark. Mean and standard deviation are reported, averaged over four random weight initializations. Other than ablation, all other model settings are held constant; thus ablations with gating removed have reduced parameter count.

Model	**Diam.**	**Ecc.**	**SSSP**
gLSTM	-0.715 $\pm$ 0.030	-4.036 $\pm$ 0.311	-2.836 $\pm$ 0.178
- Output gate	-0.70 $\pm$ 0.05	-3.71 $\pm$ 0.16	-2.77 $\pm$ 0.19
- Input gate	-0.75 $\pm$ 0.01	-4.72 $\pm$ 0.36	-3.27 $\pm$ 0.16
- Forget gate	-0.71 $\pm$ 0.03	-4.30 $\pm$ 0.21	-3.14 $\pm$ 0.07
- All gates	-0.75 $\pm$ 0.03	-4.14 $\pm$ 0.42	-3.16 $\pm$ 0.15
- K-hop aggregation	0.04 $\pm$ 0.12	0.67 $\pm$ 0.02	-3.38 $\pm$ 0.14

Table 4: Ablation of gLSTM performance on Peptides-Func and Peptides-Struct from the LRGB. Mean and standard deviation are reported, averaged over four random weight initializations. Other than ablation, all other model settings are held constant; thus ablations with gating removed have reduced parameter count.

Model	**Peptides-Func**	**Peptides-Struct**
	AP ($\uparrow$)	**MAE** ($\downarrow$)
gLSTM	0.7250 $\pm$ 0.0023	0.2527 $\pm$ 0.0015
- Output gate	0.7086 $\pm$ 0.0049	0.2540 $\pm$ 0.0016
- Input gate	0.7186 $\pm$ 0.0029	0.2524 $\pm$ 0.0027
- Forget gate	0.7236 $\pm$ 0.0063	0.2522 $\pm$ 0.0011
- All gates	0.7180 $\pm$ 0.0088	0.2526 $\pm$ 0.0012
- Positional encoding	0.7208 $\pm$ 0.0072	0.2539 $\pm$ 0.0036
- K-hop aggregation	0.6030 $\pm$ 0.0096	0.2638 $\pm$ 0.0010

Table 5: Ablation of gLSTM performance on Peptides-Func and Peptides-Struct from the LRGB. Mean and standard deviation are reported, averaged over four random weight initializations. All methods adhere to a 500k parameter limit such that hidden dimension varies to keep parameter count as close to this as possible.

Model	**Peptides-Func**	**Peptides-Struct**
	AP ($\uparrow$)	**MAE** ($\downarrow$)
gLSTM	0.7250 $\pm$ 0.0023	0.2527 $\pm$ 0.0015
- Output gate	0.7202 $\pm$ 0.0056	0.2537 $\pm$ 0.0011
- Input gate	0.7193 $\pm$ 0.0110	0.2518 $\pm$ 0.0027
- Forget gate	0.7148 $\pm$ 0.0107	0.2545 $\pm$ 0.0043
- All gates	0.7188 $\pm$ 0.0060	0.2528 $\pm$ 0.0035
- Positional encoding	0.7211 $\pm$ 0.0062	0.2601 $\pm$ 0.0017
- K-hop aggregation	0.6030 $\pm$ 0.0096	0.2638 $\pm$ 0.0010

B.3 HYPERPARAMETERS

In Tables 6 and 7 we present the hyperparameter sweeps and chosen hyperparameters for GPP and LRGB respectively.

Table 6: Hyperparameter sweeps for gLSTM on LRGB tasks. In bold are the hyperparameters that achieved the best validation set performance, and thus were those used in the main results of the paper. Note that hidden dimension was not directly swept over, as this was maximized for each configuration such that the model remained within the 500k parameter budget. Due to compute limitations, hyperparameter sweeps were not exhaustive, but used *Weights and Biases* Bayesian Optimization routine with Hyperband early termination.

Hyperparameter	**Peptides-Func**	**Peptides-Struct**
Memory Dimension	8, 16, **32**	8, **16**, 32
Number of Heads	1-**2**-8	1-**5**-8
Message Passing Layers	10-**27**-50	4-**23**-40
Input Norm Type	**Layer**	**Layer**, None
Hidden Norm Type	**Group**	**Group**
Act. Func. (between block)	GeLU, ReLU, **None**	GeLU, **ReLU**, None
Dropout	**0.1**	**0.0**, 0.1, 0.2
Hidden Dimension	**45**	**42**

Table 7: Hyperparameter sweeps for gLSTM on GPP tasks. In bold are the hyperparameters that achieved the best validation set performance, and thus were those used in the main results of the paper. Hyperparameters were tested exhaustively via grid search.

Hyperparameter	**Diam.**	**Ecc.**	**SSSP**
Memory Dimension	8, **16**	**8**, 16	8, **16**
Number of Heads	**1**, 2, 3, 4	1, 2, 3, **4**	**1**, 2, 3, 4
Message Passing Layers	1, 5, 10, **20**	1, 5, **10**, 20	1, 5, **10**, 20
Input Norm Type	**None**	**None**	**None**
Hidden Norm Type	**Group**	**Group**	**Group**
Act. Func. (between block)	Tanh, **ReLU**, None	Tanh, **ReLU**, None	**Tanh**, ReLU, None
Dropout	**0.0**	**0.0**	**0.0**
Hidden Dimension	**10**, 20, 30	10, **20**, 30	**10**, 20, 30

B.4 OVERSMOOTHING AND LONG RANGE DEPENDENCIES

We test empirically that gLSTM is able to learn long range dependencies by evaluating on the RingTransfer task introduced in Di Giovanni et al. (2023a). Results for gLSTM, GCN and GNN-SSM (Arroyo et al., 2025) are shown for various ring sizes (and corresponding number of message passing layers) in Figure 8.

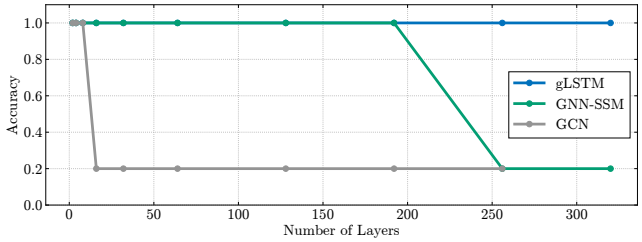


Figure 8: Performance on the RingTransfer task.

B.5 Additional NAR Classification Results

In this section, we present additional results from the NAR task presented in the main body of the paper.

We first visualize the Jacobian norms - separated by selected vs background nodes - for the mixed aggregation strategies used in the main paper in Figure 9. This is, in effect, the more granular plot of Figure 6b.

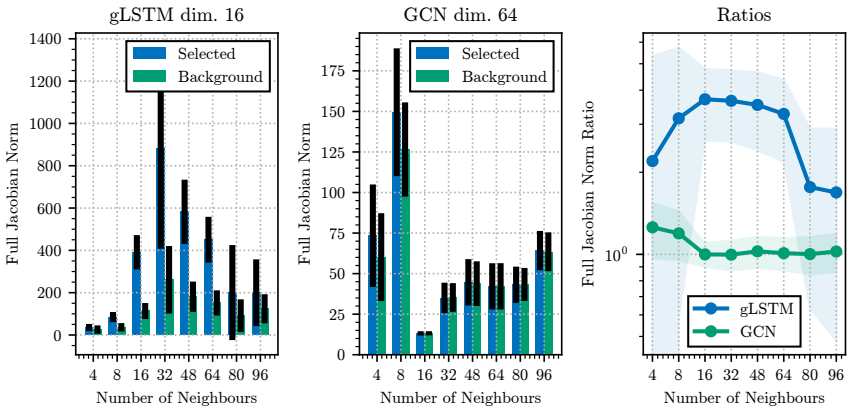


Figure 9: **Left**: Mean Jacobian norms for gLSTM of memory dimension 16, with varying number of neighbors in the NAR task, separated by whether the neighbor node corresponds to the given query (selected) or not (background). **Middle**: Same, with GCN of hidden dimension 64. **Right**: Mean ratios of Jacobian norms for selected nodes to background nodes, for these two models. Standard deviation visualized in bar chart error bars and line chart shaded area.

We next separate out no-K-hop and K-hop aggregation, and plot results for a larger set of models, in Figures 10 and 11 respectively.

We additionally verify that the number of layers is not the reason behind GCN being unable to solve NAR at higher neighbor counts. Figure 13 visualizes the performance of GCN models with hidden dimension 128 and various layer counts; it transpires that 2 layers performs best out of the tested layer counts.

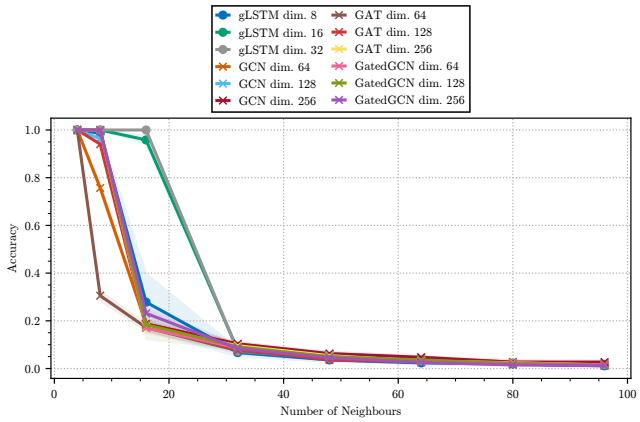


Figure 10: NAR Accuracy where all models do *not* use K-hop aggregation, for an expanded set of models.

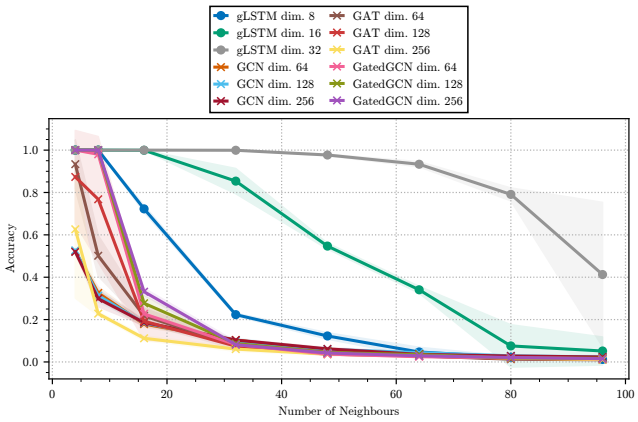


Figure 11: NAR Accuracy where all models *do* use K-hop aggregation, for an expanded set of models.

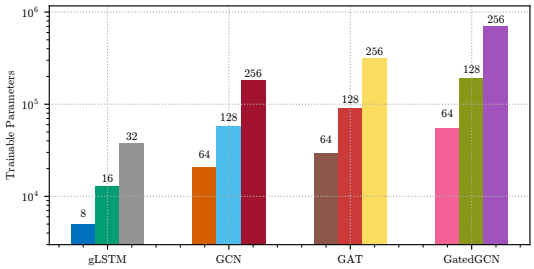


Figure 12: Number of trainable parameters for the expanded set of models tested in Figures 10 and 11.

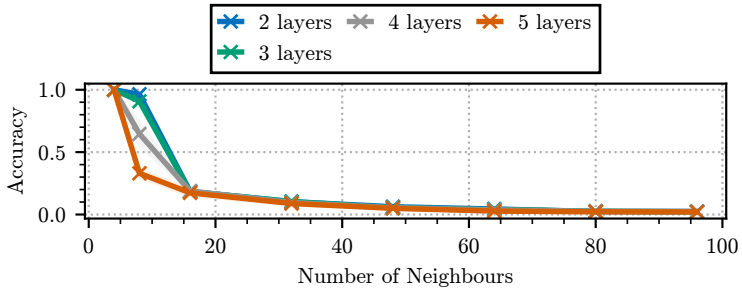
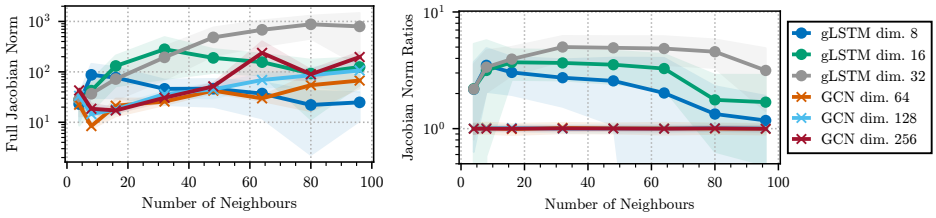


Figure 13: NAR Accuracy for GCN of hidden dimension 128, no K-hop, for varying numbers of GCN layers.

B.5.1 Additional Sensitivity Metric Results

We plot in this section the sensitivity metric trends of gLSTM vs GCN, both using K-hop aggregation.

Figure 14a visualizes the Jacobian norms for different model sizes and numbers of neighbors; Figure 14b shows the ratios between selected and background node Jacobian norms. Figure 15 separates out the Jacobian norms for gLSTM memory dimension 16 and GCN hidden dimension 64. Figure 16 visualizes the Hessian mixing metric for all models.



(a) Jacobian norms (b) Jacobian norm ratios

Figure 14: Left: Average Jacobian norms for different gLSTM and GCN models, with varying number of neighbors in the NAR task. Right: The ratio between the Jacobian norms of the selected (key corresponds to query) to background (key is different from query) neighbor nodes, for the different models - see Figure 15. This plot differs from that in the main body of the paper in that both gLSTM and GCN use K-hop aggregation.

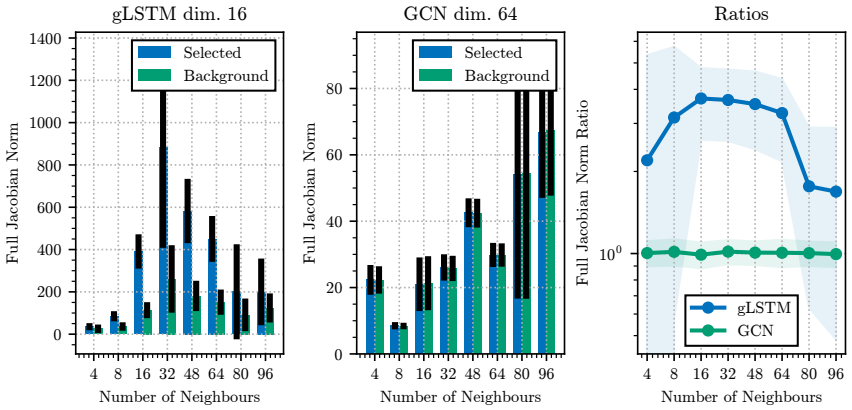


Figure 15: Left: Jacobian norms for gLSTM of memory dimension 16, with varying number of neighbors in the NAR task, separated by whether the neighbor node corresponds to the given query (selected) or not (background). Middle: Same, with GCN of hidden dimension 64. Right: Ratios of Jacobian norms for selected nodes to background nodes, for these two models. This plot differs from that in the main body of the paper in that both gLSTM and GCN use K-hop aggregation.

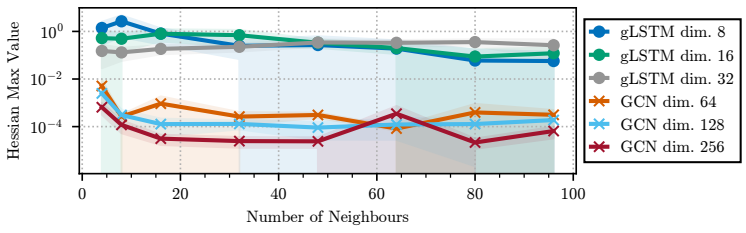


Figure 16: Mean of the maximum Hessian values for different gLSTM and GCN models, averaged across test set examples and different neighbor nodes. This plot differs from that in the main body of the paper in that both gLSTM and GCN use K-hop aggregation.

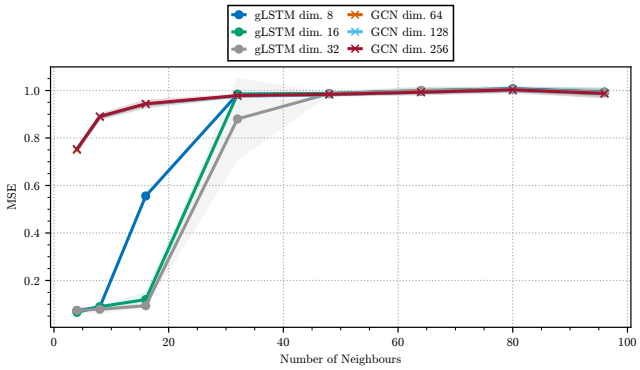


Figure 17: NARR MSE where all models do *not* use K-hop aggregation.

B.6 Neighbor Associative Recall Regression Results

In this section, we present results for the *regression* variant of the NAR task presented in the main body of the paper. We refer to this as Neighbor Associative Recall Regression (NARR).

Similarly to NAR, for a given neighborhood size N we create a graph of $N + 3$ nodes. This graph consists of N "neighbor" nodes, a central node to which they are all connected, and an intermediate node connected to the central node and a "query" node connected only to the intermediate node.

Each of the neighbor nodes has a feature vector representing a key and a value. The values consist of a fixed-dimensional vector of length V where each element is randomly sampled from a standard normal distribution. The keys are each unique one-hot vectors of dimension N. The query node's feature vector contains a single one-hot vector, equal to one of the one-hot vectors of the neighbor nodes. The target of the graph is for the central node to predict the *value* of the neighbor node, corresponding to the key that matches the query node. Each node u is therefore equipped with an input feature vector $\boldsymbol{x} \in \mathbb{R}^{V+2N}$, where the first V elements comprise the value, the next N elements the key and the final N elements the query. Where a node does not have one of these features, the vector elements are set to zero. We note the use of one-hot encoding for keys and values means that the first linear layer of the model acts as a learned embedding function, where multiplication with the one-hot encoding simply selects the corresponding column of the weight matrix. For our experiments, we use $V = 16$.

Since the value vectors lack the sparsity of NAR, this appears to be a "harder" task in the sense that it is more taxing on memory capacity. This means that some of the over-squashing trends are more defined, particularly trends in sensitivity-based measures - see Appendix B.6.1. However, our experiments suggest that the regression target means that NARR becomes too hard for vector-memory MPNNs to effectively solve, visible in Figures 17 and 18.

Performance (MSE) for NARR is shown in Figures 17 and 18 for no-K-hop and K-hop aggregation respectively. We note that the performance curves in Figure 18 look similar to those obtained by the sequence modeling variant of this experiment in Schlag et al. (2021). The number of trainable parameters is shown in Figure 19.

B.6.1 Relationship to Over-Squashing Sensitivity Metrics

As with NAR in the main paper, we visualize the behavior of sensitivity-based over-squashing metrics for different neighbor counts and different models. Similarly to the main paper, we compare gLSTM using K-hop aggregation and GCN without. We note that – perhaps due to the increased difficulty of the task – the trends discussed in Section 5.2 are actually *more* pronounced for the NARR task.

Figure 20a visualizes the Jacobian norms for different model sizes and numbers of neighbors; Figure 20b shows the ratios between selected and background node Jacobian norms. Figure 21 separates out the Jacobian norms for gLSTM memory dimension 16 and GCN hidden dimension 64. Figure 22 visualizes the Hessian mixing metric for all models.

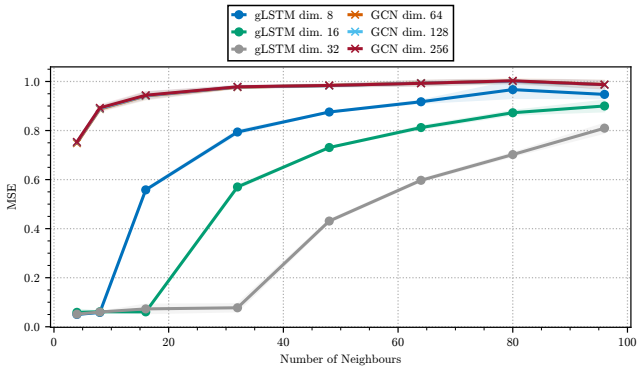


Figure 18: NAR Accuracy where all models *do* use K-hop aggregation.

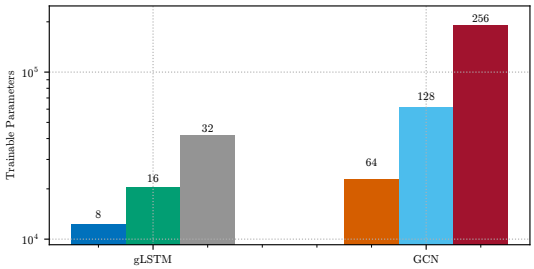
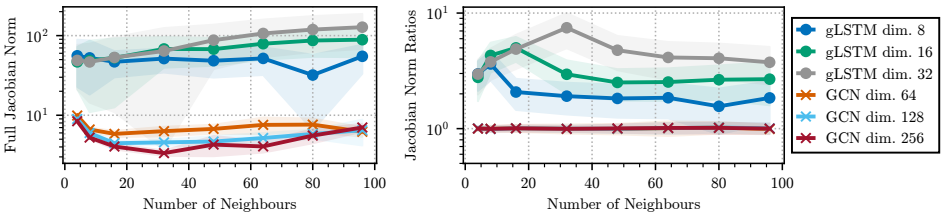


Figure 19: Number of trainable parameters for the expanded set of models tested in Figures 17 and 18.

We note that the sensitivity difference between selected and background nodes is particularly stark here, even more so for classification-based NAR; gLSTM consistently shows a sharp drop-off in Figure 20b at the memory dimension, and GCN maintains a ratio remarkably close to unity. This closely aligns with the performance of these models, Figure 18 demonstrates that gLSTM performance begins to drop off quickly when the number of neighbors matches the memory dimension, and Figure 17 demonstrates that GCN is never able to solve the task, for any tested number of neighbors.

We hypothesize that the strong impact of the K-hop aggregation on the model's ability to selectively recall - particularly visible for NARR - may partially explain the dramatic performance decrease when ablating this aggregation, discussed in Appendix B.2. We note that, while gLSTM consistently demonstrates superior performance to GCN, the improved performance is most striking when additionally using K-hop aggregation; it appears that the inductive bias introduced by the K-hop aggregation is particularly suited to the selective recall required by this task.



(a) Jacobian norms (b) Jacobian norm ratios

Figure 20: Left: Average Jacobian norms for different gLSTM and GCN models, with varying number of neighbors in the NARR task. Right: The ratio between the Jacobian norms of the selected (key corresponds to query) to background (key is different from query) neighbor nodes, for the different models - see Figure 21.

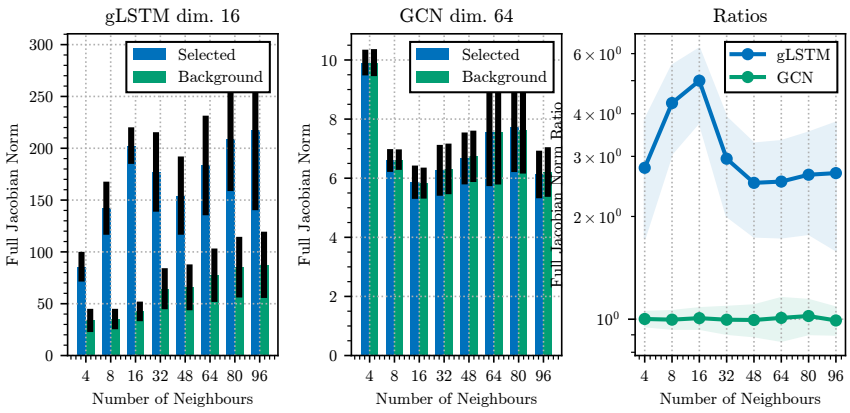


Figure 21: Left: Jacobian norms for gLSTM of memory dimension 16, with varying number of neighbors in the NARR task, separated by whether the neighbor node corresponds to the given query (selected) or not (background). Middle: Same, with GCN of hidden dimension 64. Right: Ratios of Jacobian norms for selected nodes to background nodes, for these two models.

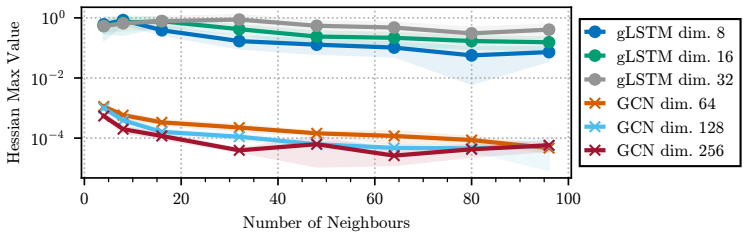


Figure 22: Mean of the maximum Hessian values for different gLSTM and GCN models, averaged across test set examples and different neighbor nodes.