ReHub: Linear Complexity Graph Transformers with Adaptive Hub-Spoke Reassignment

Anonymous authors
Paper under double-blind review

Abstract

We present ReHub, a novel graph transformer architecture that achieves linear complexity through an efficient reassignment technique between nodes and virtual nodes. Graph transformers have become increasingly important in graph learning for their ability to utilize long-range node communication explicitly, addressing limitations such as oversmoothing and oversquashing found in message-passing graph networks. However, their dense attention mechanism scales quadratically with the number of nodes, limiting their applicability to large-scale graphs. ReHub draws inspiration from the airline industry's hub-and-spoke model, where flights are assigned to optimize operational efficiency. In our approach, graph nodes (spokes) are dynamically reassigned to a fixed number of virtual nodes (hubs) at each model layer. Recent work, Neural Atoms (Li et al., 2024), has demonstrated impressive and consistent improvements over GNN baselines by utilizing such virtual nodes; their findings suggest that the number of hubs strongly influences performance. However, increasing the number of hubs typically raises complexity, requiring a trade-off to maintain linear complexity. Our key insight is that each node only needs to interact with a small subset of hubs to achieve linear complexity, even when the total number of hubs is large. To leverage all hubs without incurring additional computational costs, we propose a simple yet effective adaptive reassignment technique based on hub-hub similarity scores, eliminating the need for expensive node-hub computations. Our experiments on long-range graph benchmarks indicate a consistent improvement in results over the base method, Neural Atoms, while maintaining a linear complexity instead of $O(n^{3/2})$. Remarkably, our sparse model achieves performance on par with its non-sparse counterpart. Furthermore, ReHub outperforms competitive baselines and consistently ranks among the top performers across various benchmarks.

1 Introduction

Learning on graphs is essential in numerous domains, including social networks for influence prediction, biological networks for understanding protein interactions, molecular graphs for predicting chemical properties, knowledge graphs for recommendation systems, and financial networks for fraud detection and risk assessment. Graph neural networks (GNNs) have emerged as powerful tools in these areas, operating via message passing between connected nodes. However, a significant challenge with GNNs is their limited communication range. While stacking message-passing layers can increase the communication distance, it comes at a computational cost and can cause issues such as oversmoothing and oversquashing (Alon & Yahav, 2020; Topping et al., 2021).

Inspired by the success of transformers in natural language processing (Vaswani et al., 2017), graph transformers offer a solution by enabling global node communication through attention mechanisms (Dwivedi & Bresson, 2020; Shehzad et al., 2024). This overcomes the communication bottlenecks of GNNs, but it comes at a significant computational cost. The quadratic complexity of dense attention operations limits the scalability of graph transformers, as even modest-sized graphs can exhaust GPU memory. Several methods have been suggested to reduce the complexity of global attention. For example, GraphGPS (Rampášek et al., 2022) combines sparse attention mechanisms like Performer (Choromanski et al., 2020) or Big Bird

(Zaheer et al., 2020). Originally designed for processing sequences rather than graph structure, these linear-memory transformers induce significant computational time and do not match the performance of dense attention (Shirzad et al., 2023).

Recently, transformer-based graph networks have utilized the addition of virtual global nodes, through which graph nodes communicate to sparsify the attention. By constraining the attention to be between the graph nodes and these virtual nodes, the attention complexity is reduced to the number of nodes times the number of virtual nodes, allowing the overall complexity to be governed by the number of virtual nodes. Exphormer (Shirzad et al., 2023) maintains linear complexity by using a fixed number of virtual nodes, while Neural Atoms (Li et al., 2024) explores both a fixed number and a ratio relative to the number of nodes. An important finding in Neural Atoms is that adding more virtual nodes increases prediction accuracy, creating a trade-off between computational complexity and accuracy.

In this work, we introduce ReHub, a novel graph transformer architecture that achieves linear complexity by dynamically reassigning graph nodes to virtual nodes. We are inspired by complex systems where efficient connectivity and adaptability are crucial for optimal performance. A pertinent example is the airline industry, where flights are dynamically assigned to a limited number of major airports (hubs) to optimize operational efficiency. We identify the graph nodes with spokes and the virtual nodes with hubs. The key insight of our approach stems from noting that the transformer's complexity is driven by spoke-hub attention. Therefore, it is not necessary to reduce the total number of hubs to achieve linear complexity; instead, using a fixed, small number of connected hubs per spoke is sufficient. To effectively utilize all hubs without increasing computational cost, we introduce a simple yet efficient adaptive reassignment mechanism. In order to do so, and yet avoid the costly computation of the entire set of spoke-hub interactions, our reassignment mechanism is based on hub-hub similarity scores, which are cheap to compute.

In summary, then, our primary contribution is a novel graph transformer architecture that integrates global attention with an efficient spoke-hub reassignment strategy, significantly enhancing scalability while maintaining performance. Our experiments on long-range graph benchmarks indicate a consistent improvement in results over the base method, Neural Atoms, while keeping a linear complexity instead of $O(n^{3/2})$. Remarkably, our sparse model achieves performance on par with its non-sparse counterpart. Furthermore, ReHub outperforms competitive baselines and consistently ranks among the top performers across various benchmarks.

2 Related work

Learning on large graphs Graph learning architectures are a well-established and highly active field of research (Wu et al., 2020). Common GNNs, such as GCN/GCN2 (Kipf & Welling, 2016; Chen et al., 2020), GAT/GATv2 (Velickovic et al., 2017; Brody et al., 2021), GIN/GINE (Xu et al., 2018; Hu et al., 2019), and GatedGCN (Bresson & Laurent, 2017), rely on a message-passing architecture that aggregates information. In each layer every graph node updates its representation by aggregating the neighboring nodes. This architecture inherently limits their ability to accumulate information over large distances due to phenomena such as over-smoothing (Alon & Yahav, 2020), where node representations become indistinguishable, and over-squashing (Topping et al., 2021) , where the capacity to propagate information is restricted by bottlenecks. Consequently, learning on large graphs remains a persistent challenge (Duan et al., 2022). To address this issue, some methods focus on reducing the memory footprint. One approach involves dividing graphs into mini-batches (Wu et al., 2024), while another uses only segments of the graph for training (Cao et al., 2024).

Transformer architectures have recently gained popularity for graph-based tasks (Müller et al., 2023). These methods address the over-smoothing and over-squashing issues by enabling all nodes to interact with each other through attention (Velickovic et al., 2017; Ying et al., 2021). However, this approach is computationally inefficient, with quadratic time and memory consumption in the number of nodes. To mitigate these inefficiencies, more efficient transformer architectures have emerged which utilize different approaches to reduce the number of computations such as approximations in Performer (Choromanski et al., 2020), predefined attention patterns in BigBird (Zaheer et al., 2020) and better parallelism and partitioning in FlashAttention (Dao et al., 2022). GraphGPS (Rampášek et al., 2022) proposes a general framework for combining message-passing neural networks (MPNNs) with attention at each layer, facilitating the use of such attention mechanisms.

Recent works have introduced approaches that leverage the structure and inherent information of the graph, such as tokenization of hops of node's neighbors in NAGphormer and applying structure-preserving attention on encoded sequences of sub-graphs in Gophormer (Chen et al., 2022; Zhao et al., 2021). Some of these works ensure that the computational complexity remains linear in the number of nodes (Shirzad et al., 2023; Chen et al., 2022; Wu et al., 2024; 2022) which allows them to be applied to larger graphs. We also propose a linear complexity architecture that passes nodes' information through a medium that efficiently orchestrates the passage of information in the graph and between nodes.

Finally, we note that recently, State Space Models (SSMs) (Gu & Dao, 2023) have emerged as a promising approach for efficiently processing large sequences, with their adaptation to graphs showing notable results (Wang et al., 2024; Behrouz & Hashemi, 2024). However, in this work, we focus on transformer-based architectures, which remain the current go-to approach.

Virtual nodes The concept of virtual nodes involves introducing new external nodes that interact with the graph to facilitate information exchange between existing nodes (Gilmer et al., 2017). Recent studies (Shirzad et al., 2023; Cai et al., 2023; Hwang et al., 2022) utilize this concept to extend the graph's capability to capture long-range information through message-passing. Other research explores the integration of virtual nodes within the context of transformers (Shirzad et al., 2023; Li et al., 2024; Fu et al., 2024). Similarly, we use virtual nodes that communicate with both each other and the graph nodes through attention. However, unlike prior methods, we dynamically update the connectivity between the virtual nodes and graph nodes at each layer, enhancing the flow of information.

3 Method

3.1 Pipeline overview

Our proposed architecture, depicted in Figure 1, is aimed at handling long-range graph node communication while maintaining computational efficiency. This is implemented through hubs, that act as information aggregators and distributors from and to the spokes. This allows for long range communication to be manifested as hub-hub communication. ReHub is carefully designed to follow our key observation that the complexity can be kept linear as long as: (1) the number of hubs N_h is kept small enough, on the order of $\sqrt{N_s}$, where N_s is the number of spokes; and (2) k, the number of hubs connected to each spoke per layer, is a small constant (*e.g.*, $k = 3$).

In what follows we describe the architecture of ReHub, define every component and the interaction between spokes and hubs. First we present an overview of the notation. Then we present an initialization scheme for the hubs and explain each part of the architecture: (1) Spokes-Spokes update (2) Spokes-Hubs update (3) Hubs-Hubs update (4) Hubs-Spokes update (5) Hub (Re)Assignment. Finally, we show that the complexity given by this architecture is linear in the number of nodes.

3.2 Notation

Spokes and Hubs. Throughout this paper, we refer to the graph nodes as "spokes" [1] and the added virtual nodes as "hubs". The number of spokes is N_s, and they are indexed with $i_s = 1, \ldots, N_s$. Each spoke has features represented by $\boldsymbol{s}_{i_s} \in \mathbb{R}^d$, with the collection of all spoke features denoted by $\boldsymbol{s}$. Similarly, the number of hubs is denoted by N_h, and they are indexed by $i_h = 1, \ldots, N_h$. Each hub has features represented by $\boldsymbol{h}_{i_h} \in \mathbb{R}^d$, and the collection of all hub features is denoted by $\boldsymbol{h}$. The binary matrix $\boldsymbol{E} \in \{0,1\}^{N_s \times N_h}$, referred to as the *hub assignment matrix*, indicates which spokes are connected to which hubs:

$$\boldsymbol{E}_{i_s,i_h} = \begin{cases} 1 & \text{if spoke } i_s \text{ is connected to hub } i_h \\ 0 & \text{otherwise} \end{cases} \quad \text{s.t.} \quad \boldsymbol{E}\,\mathbf{1}_{N_h} = k \cdot \mathbf{1}_{N_s}, \tag{1}$$

[1] In this work, we use "spokes" to represent graph nodes, deviating from the more common usage where "spokes" refer to edges.

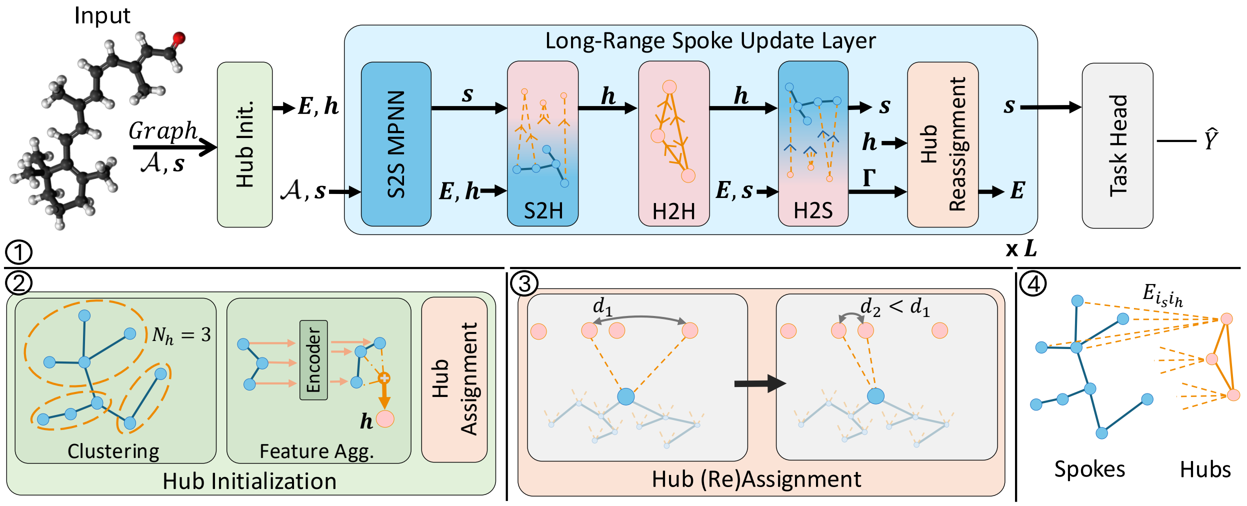

Figure 1: Illustration of ReHub architecture. (1) Overview of the different steps in the architecture. $\mathcal{A}$ is the input (spoke) graph's adjacency matrix; spoke features $\boldsymbol{s}$; hub features $\boldsymbol{h}$; hub assignment $\boldsymbol{E}$; and $\boldsymbol{\Gamma}$ contains attention scores from the Hubs-Spokes attention. (2) Hubs initialization. Spokes are first clustered, then each cluster is aggregated to compute hub features. Finally, each spoke is assigned more hubs. (3) Hub (Re)Assignment. Each spoke selects the k hubs closest to its most similar hub as its new assignment. (4) Illustration of connectivity between spokes and hubs. Information pass between spokes with an MPNN; while the interaction between hubs and spokes is performed via attention and only through available connections $E_{i_s i_h}$. The hubs pass information between themselves via full self-attention.

where $\mathbf{1}_N$ denotes an all-ones column vector of length N. Namely, a given spoke is connected to exactly k hubs, where $k = O(1)$. $\boldsymbol{E}$ can be implemented as a sparse assignment matrix. Finally, where relevant the network layer is denoted using a superscript.

Bipartite graph attention. In our pipeline we utilize graph attention (Brody et al., 2021) between spokes and hubs that interchangeably act as source and target graphs. The definition is as follows:

$$\boldsymbol{O}\,,\,\boldsymbol{\Gamma} = \texttt{Attention}\left(\boldsymbol{K}, \boldsymbol{Q}, \boldsymbol{E}\right), \tag{2}$$

where the input consists of source graph nodes $\boldsymbol{K} \in \mathbb{R}^{n_k \times d}$, destination graph nodes $\boldsymbol{Q} \in \mathbb{R}^{n_q \times d}$, and a hub assignment $\boldsymbol{E} \in \{0, 1\}^{n_k \times n_q}$ containing the connections between the nodes of the source graph with those of the destination graph. The outputs are the per-node updated features $\boldsymbol{O} \in \mathbb{R}^{n_q \times d}$ of the destination graph. We optionally output the sparse attention scores $\boldsymbol{\Gamma} \in \mathbb{R}^{n_k \times n_q}$, computed for the non-zero entries of $\boldsymbol{E}$. For a non-bipartite graph the same formalism can be applied by taking $\boldsymbol{K} = \boldsymbol{Q}$, enabling self-attention within the graph.

3.3 Initialization

We initiate ReHub by creating $N_h = r\sqrt{N_s}$ hubs, where r is the hub-ratio, set to 1 in almost all benchmarks. To populate the hubs features with meaningful values we compute them based on (i) clustering the input graph (*i.e.* the graph of spokes), specified by the adjacency matrix $\mathcal{A}$, and (ii) aggregating the spoke features $\boldsymbol{s}$. We then assign each spoke to k hubs. This process is illustrated in module (2) of Figure 1.

Clustering. We partition the graph $\mathcal{A}$, along with its spoke features $\boldsymbol{s}$, into N_h clusters using METIS (Karypis & Kumar, 1998). This method takes as input an adjacency matrix between spokes $\mathcal{A}$ as well as the desired number of clusters N_h; and returns as output a cluster index for each spoke. We denote each cluster as $\mathcal{C}_{i_h}$, where a spoke belongs to a cluster if $i_s \in \mathcal{C}_{i_h}$.

Hub features. For each cluster $\mathcal{C}_{i_h}$, we compute the initial hub features. For hub-cluster i_h, we aggregate the spoke features as follows:

$$\boldsymbol{h}^0_{i_h} = \texttt{Aggregate-Feat}(\{\boldsymbol{s}^0_{i_s}\}_{i_s \in \mathcal{C}_{i_h}}) \tag{3}$$

There are several options for the `Aggregate-Feat` function. For categorical or ordinal variables (*e.g.*, , atom type), one may compute a histogram. For continuous variables, one may compute the average. In our case, we choose to average the features after the nodes have passed through a positional encoding layer followed by a feedforward layer.

In Section 4.3, we compare our initialization scheme with those proposed in previous works and demonstrate that it significantly improves performance.

Hub assignment. We aim to connect each spoke with a total of k hubs. Since each spoke $\boldsymbol{s}_{i_s}$, already has a single connection to a hub $\boldsymbol{h}_{i_h}$ as a result of the clustering, we assign the remaining $k-1$ hub connections for each spoke, by selecting the hubs closest to $\boldsymbol{h}_{i_h}$ in terms of feature similarity. The assignment procedure is detailed in Section 3.4. This step results in the spoke-to-hub initial assignment matrix $\boldsymbol{E}^0_{s\to h}$.

3.4 Long-range spoke update layer

A key component of ReHub's architecture is the **long-range spoke update layer**, which leverages both the spoke graph and the connections between spokes and hubs. Specifically, to maintain efficiency, we avoid global spoke-to-spoke attention operations by using local message passing and global spoke-to-hub operations. We further keep the spoke-to-hub attention efficient by restricting the hub connectivity of spokes to a small constant number of hubs, k. Long-range information flow between spokes occurs through hub-to-hub communication, which is made efficient by selecting a number of hubs proportional to the square root of the number of spokes. This relaxes the restriction imposed by previous graph transformer methods (Shirzad et al., 2023), which limited the number of hubs to a small constant. This layer is repeated L times throughout the network. Each layer ℓ consists of five steps, as shown in Figure 1, which we describe in detail below.

(1) Spokes → Spokes: For the local spoke update using the neighboring nodes in the graph:

$$\boldsymbol{s}^{\ell+\frac{1}{2}} = \texttt{MPNN}(\boldsymbol{s}^\ell), \tag{4}$$

where `MPNN` is a single layer of a message-passing neural network. Since this operation is restricted to the 1-ring neighborhood, it remains efficient. Notably, any type of message-passing scheme can generally be integrated into our pipeline.

(2) Spokes → Hubs: Given the updated spokes, we update the hubs using the assignment matrix. Since each spoke is connected to exactly k hubs, the operation is sparse and memory efficient.

$$\boldsymbol{h}^{\ell+\frac{1}{2}} = \texttt{Attention}(\boldsymbol{s}^{\ell+\frac{1}{2}}, \boldsymbol{h}^\ell, \boldsymbol{E}^\ell_{s\to h}) \tag{5}$$

(3) Hubs → Hubs: The hubs now interact using self-attention. Even though the connectivity $\boldsymbol{E}_{full}$ is dense, the overall number of hubs is kept on the order of $\sqrt{N_s}$, ensuring overall linear complexity.

$$\boldsymbol{h}^{\ell+1} = \texttt{Attention}(\boldsymbol{h}^{\ell+\frac{1}{2}}, \boldsymbol{h}^{\ell+\frac{1}{2}}, \boldsymbol{E}_{full}) \tag{6}$$

(4) Hubs → Spokes: Given the hubs, we update the spokes:

$$\boldsymbol{s}^{\ell+1}, \boldsymbol{\Gamma}^{\ell+1} = \texttt{Attention}(\boldsymbol{h}^{\ell+1}, \boldsymbol{s}^{\ell+\frac{1}{2}}, \boldsymbol{E}^\ell_{h\to s}) \qquad \boldsymbol{\Gamma}^{\ell+1} \text{ is } N_s \times N_h \tag{7}$$

where the matrix $\boldsymbol{E}^\ell_{h\to s} = \left(\boldsymbol{E}^\ell_{s\to h}\right)^T$, assuring the same efficiency as the Spokes → Hubs step.

(5) Hub (Re)Assignment: While restricting each spoke to connect with k hubs maintains efficiency, for the method to achieve its full potential, it should utilize all available hubs. We achieve this by keeping only k connected hubs per spoke at each layer, while allowing each spoke to reassign $k-1$ of its hubs before proceeding to the next layer[2]. The reassignment is based on spoke-hub similarity. Selecting the hubs closest to each spoke, however, would require $N_s \times N_h$ computations. To avoid this, we leverage the distances between hubs, which are efficient to compute.

We then retain the hub most similar to each spoke from the sparse set of connected hubs, replacing the remaining $k-1$ hubs with those closest to it. This procedure is outlined in Algorithm 1. The matrix of distances between all hub features is denoted by $\mathbf{\Delta}$. Naturally, a hub is closest to itself and would be selected first. Additionally, we use $\mathbf{\Gamma}$, the attention score matrix, to identify, for each spoke, the most similar hub to which it is connected.

Algorithm 1 Hub (Re)Assignment

```
Require: Hub-Spoke cross-attention score matrix $\mathbf{\Gamma}^{\ell+1}$ and Hub-Hub distance matrix $\mathbf{\Delta}^{\ell+1}$
  for $i_h = 1$ to $N_h$ do
    $\mathcal{H}(i_h) = \texttt{Bottom-k-Indices}(\text{row } i_h \text{ of } \mathbf{\Delta}^{\ell+1})$
  end for
  for $i_s = 1$ to $N_s$ do
    $i_h^* = \arg\max_{i_h} \mathbf{\Gamma}^{\ell+1}_{i_s i_h}$
    $\boldsymbol{E}^{\ell+1}_{i_s i_h} = \begin{cases} 1 & \text{if } i_h \in \mathcal{H}(i_h^*) \\ 0 & \text{otherwise} \end{cases}$
  end for
  return $\boldsymbol{E}^{\ell+1}$
```

Final prediction. The pipeline concludes with a task-specific prediction head. In this work, we demonstrate tasks such as graph classification, regression, node classification, and link prediction. These prediction heads use an MLP on the final spoke feature predictions, $\boldsymbol{s}^L$, with further aggregation for graph-level tasks. For link prediction, the pipeline computes a similarity score for every pair of nodes connected by an edge.

Complexity Recall that sparse attention is used, where multiplications are performed only between spokes and the k hubs to which they are connected. The resulting time and memory complexity for each Spokes-to-Hubs interaction step is $O(N_s k)$, and for the Hubs self-attention step, it is $O(N_h^2)$. By taking $N_h = O(\sqrt{N_s})$ and $k = O(1)$, we achieve linear complexity in the total number of spokes N_s, as desired.

4 Experiments

Methods in comparison. We compare ReHub against leading graph transformer based methods. GraphGPS (Rampášek et al., 2022) offers a framework to integrate MPNNs of different types (Kipf & Welling, 2016; Chen et al., 2020; Hu et al., 2019; Bresson & Laurent, 2017) and transformers. Transformer indicates a straightforward adaptation of the standard transformer architecture Vaswani et al. (2017) to graphs. Spectral Attention Networks (SAN) (Kreuzer et al., 2021) employ attention on the fully connected graph in addition to graph attention using the original edges. Closest to our method is Neural Atoms (Li et al., 2024) which utilizes a set of different, learned, virtual nodes at each layer. Neural Atoms is able to propagate long-range information, improve performance across various tasks and can be modularly applied to different MPNNs. As opposed to Neural Atoms, in this work we aim to tackle the efficiency aspect, *i.e.* to maintain linear complexity in the number of nodes in a graph without a loss of performance. Exphormer (Shirzad et al., 2023) introduces a transformer architecture that achieves linear computational complexity by leveraging expander graphs (Alon, 1986) to define sparse attention patterns. In this model, each node attends only to a fixed number of neighbors specified by a fixed expander graph, and a few global virtual nodes connected to all nodes are used to capture global context. In contrast, our proposed method ReHub employs a dynamic model where virtual nodes (hubs) are connected to subsets of nodes (spokes) rather than to all nodes. ReHub allows for rewiring connections between layers, enhancing adaptability while avoiding bottlenecks associated with fully connected global nodes, all while maintaining linear computational complexity.

[2] We note that, although we replace $k-1$ hubs while keeping the closest hub connected, in later layers, that closest hub may change. This flexibility prevents us from being overly constrained by the initial hub selection.

Datasets. Due to lack of large scale datasets that explicitly benchmark long-range capabilities we divide the evaluation to (1) long-range communication ability and (2) large graphs to verify memory efficiency. For long-range communication we evaluate ReHub on the long-range graph benchmark (LRGB) which is is widely used to evaluate methods which aim at overcoming issues such as oversmoothing and oversquashing. LRGB comprises five datasets. Two of the datasets are image-based graph datasets: PascalVOC-SP and COCO-SP which contain superpixel graphs of the well known image segmentation datasets PascalVOC and COCO. The latter three datasets are molecular datasets: Peptides-Func, Peptides-Struct and PCQM-Contact, which require the prediction of molecular interactions and properties that require global aggregation of information. For evaluation on large graphs we show results on graph datasets of citation networks: OGBN-Arxiv and Coauthor Physics which include about 170K and 30K nodes respectively, with the task of node class prediction. Additionally, we evaluate peak memory consumption on a custom dataset of large random regular graphs (Steger & Wormald, 1999; Kim & Vu, 2003) of gradually increasing sizes from 10K to 700K nodes. Additional statistics about the datasets is available in Appendix A.1

Metrics and evaluation. We evaluate our models using several metrics: Average Precision (AP), Mean Absolute Error (MAE), Mean Reciprocal Rank (MRR), F1 Score, and Accuracy. These are standard metrics and we refer to Dwivedi et al. (2022) for more details. For the evaluation of ReHub, we report the mean ± std over 5 runs, each with a different random seed. Results for other models, including NeuralAtoms (Li et al., 2024), Exphormer (Shirzad et al., 2023) and GraphGPS (Rampášek et al., 2022), were reported as published in their original works.

Hardware. All experiments were performed using one NVIDIA L40 GPU with 48GB of memory.

4.1 Long-range graph benchmark

A major challenge in graph learning is long range communication – scenarios where the prediction relies on information residing at far location of the graph. In this experiment, we evaluate ReHub on a set of such tasks provided by the LRGB dataset. We split the comparison in two, first establishing the modularity of ReHub by integrating it with various MPNN layers; we then follow with a comparison against leading methods.

MPNN modularity. Similar to Neural Atoms, which improves performance across various MPNNs, ReHub is equally modular. In Table 1, we present a comparison of ReHub using several common MPNNs. Results are shown for both the sparse case—where the number of hubs connected to each spoke per layer, k, is small—and a dense variant, ReHub-FC, where each spoke is fully connected to all hubs. The performance of both the sparse and dense configurations is compared to Neural Atoms as well the vanilla MPNN technique, demonstrating significant improvements across datasets and MPNNs. Interestingly, we observe that the performance of Neural Atoms is strongly affected by the base MPNN used. *e.g.*, for Peptides-func the final AP ranges between 0.60 and 0.66, while ReHub demonstrates increased robustness ranging between 0.64 and 0.67. Remarkably, thanks to our reassignment procedure, which promotes high utilization of all hubs, our sparse version achieves performance comparable to the dense version.

Comparison with baselines. In Table 2 we present a comparison between our ReHub using the best performing MPNN and state of the art methods on the LRGB benchmark. As can be seen, ReHub consistently scores among the top two methods.

4.2 Performance on large graphs

Peak memory vs. graph size. Our method demonstrates linear memory complexity. To showcase this in practice, we compare the peak memory consumption of our approach to other methods on graphs of varying sizes. Since no existing benchmark offers a collection of gradually growing graph sizes we instead construct a series of toy graphs with sizes varying between 10K and 700K. The graphs are d-regular (*i.e.* each node has d neighbors), and are populated with random node features and edge attributes. In this experiment, we set $d = 3$. To keep the comparison fair, for all methods we used similar parameters like the number of layers and hidden dimension. A detailed description of the used parameters is provided in Appendix A.2. For Neural Atoms, we follow the guidelines provided in the paper and use a ratio of 0.1 for the number of

Table 1: **MPNN modularity**. Test performance on datasets from the long-range graph benchmarks (LRGB) (Dwivedi et al., 2022) compared on various GNN types to Neural Atoms (Li et al., 2024). ReHub-FC has each spoke fully connected to all hubs. Best results are colored: **first**, **second**.

Model	Peptides-func	Peptides-struct	PCQM-Contact
	AP ↑	MAE ↓	MRR ↑
GCN	0.5930 ± 0.0023	0.3496 ± 0.0013	0.2329 ± 0.0009
+ NeuralAtoms	0.6220 ± 0.0046	0.2606 ± 0.0027	0.2534 ± 0.0200
+ ReHub-FC	**0.6663 ± 0.0053**	**0.2489 ± 0.0011**	**0.3492 ± 0.0012**
+ ReHub	**0.6656 ± 0.0043**	**0.2497 ± 0.0021**	**0.3469 ± 0.0014**
GCN2	0.5543 ± 0.0078	0.3471 ± 0.0010	0.3161 ± 0.0004
+ NeuralAtoms	0.5996 ± 0.0033	0.2563 ± 0.0020	0.3049 ± 0.0006
+ ReHub-FC	**0.6427 ± 0.0085**	**0.2511 ± 0.0015**	**0.3386 ± 0.0026**
+ ReHub	**0.6406 ± 0.0030**	**0.2530 ± 0.0029**	**0.3375 ± 0.0013**
GINE	0.5498 ± 0.0079	0.3547 ± 0.0045	0.3180 ± 0.0027
+ NeuralAtoms	0.6154 ± 0.0157	0.2553 ± 0.0005	0.3126 ± 0.0021
+ ReHub-FC	**0.6682 ± 0.0098**	**0.2506 ± 0.0012**	**0.3426 ± 0.0014**
+ ReHub	**0.6582 ± 0.0095**	**0.2514 ± 0.0056**	**0.3429 ± 0.0014**
GatedGCN	0.5864 ± 0.0077	0.3420 ± 0.0013	0.3218 ± 0.0011
+ NeuralAtoms	0.6562 ± 0.0075	0.2585 ± 0.0017	0.3258 ± 0.0003
+ ReHub-FC	**0.6732 ± 0.0107**	**0.2501 ± 0.0034**	**0.3526 ± 0.0014**
+ ReHub	**0.6685 ± 0.0074**	**0.2512 ± 0.0018**	**0.3534 ± 0.0014**
GatedGCN+RWSE	0.6069 ± 0.0035	0.3357 ± 0.0006	0.3242 ± 0.0008
+ NeuralAtoms	0.6591 ± 0.0050	0.2568 ± 0.0005	0.3262 ± 0.0010
+ ReHub-FC	**0.6690 ± 0.0025**	**0.2490 ± 0.0075**	**0.3523 ± 0.0012**
+ ReHub	**0.6653 ± 0.0054**	**0.2488 ± 0.0017**	**0.3528 ± 0.0008**

Table 2: Test performance on datasets from the long-range graph benchmarks (LRGB) (Dwivedi et al., 2022) compared to baselines. For Neural Atoms we show only available results. ReHub-FC has each spoke fully connected to all hubs. Best results are colored: **first**, **second**.

Model	Peptides-func	Peptides-struct	PCQM-Contact	PascalVOC-SP
	AP ↑	MAE ↓	MRR ↑	F1 Score ↑
Transformer+LapPE	0.6326 ± 0.0126	0.2529 ± 0.0016	0.3174 ± 0.0020	0.2694 ± 0.0098
SAN+LapPE	0.6384 ± 0.0121	0.2683 ± 0.0043	0.3350 ± 0.0003	0.3230 ± 0.0039
GraphGPS	0.6535 ± 0.0041	0.2500 ± 0.0005	0.3337 ± 0.0006	0.3748 ± 0.0109
Exphormer	0.6527 ± 0.0043	**0.2481 ± 0.0007**	**0.3637 ± 0.0020**	**0.3975 ± 0.0037**
NeuralAtoms	0.6591 ± 0.0050	0.2553 ± 0.0005	0.3262 ± 0.0010	n/a
ReHub-FC (Ours)	**0.6732 ± 0.0107**	0.2489 ± 0.0011	0.3526 ± 0.0014	0.3526 ± 0.0045
ReHub (Ours)	**0.6685 ± 0.0074**	**0.2488 ± 0.0017**	**0.3534 ± 0.0014**	**0.3860 ± 0.0172**

virtual nodes. As this ratio results in an asymptotic complexity of $O(N_s^2)$ we additionally include results for Neural Atoms with $N_h = \sqrt{N_s}$ for a more memory efficient version reaching $O(N_s^{3/2})$. For Exphormer, the expander graph has a degree of 3, which is the same value as k used for ReHub. The results, shown in Figure 2, indicate that our method uses less than half the memory of other methods while exhibiting a linear memory usage trend.

Memory consumption and accuracy on large graphs benchmarks. We evaluate ReHub on the competitive Coauthor Physics and OGBN-Arxiv datasets which have about 35K and 170K nodes respectively. Comparing ReHub to Exphormer on Table 3 shows an improved memory consumption by about 36% for Coauthor Physics and 13% for OGBN-Arxiv while accuracy is comparable. We follow Exphormer and include

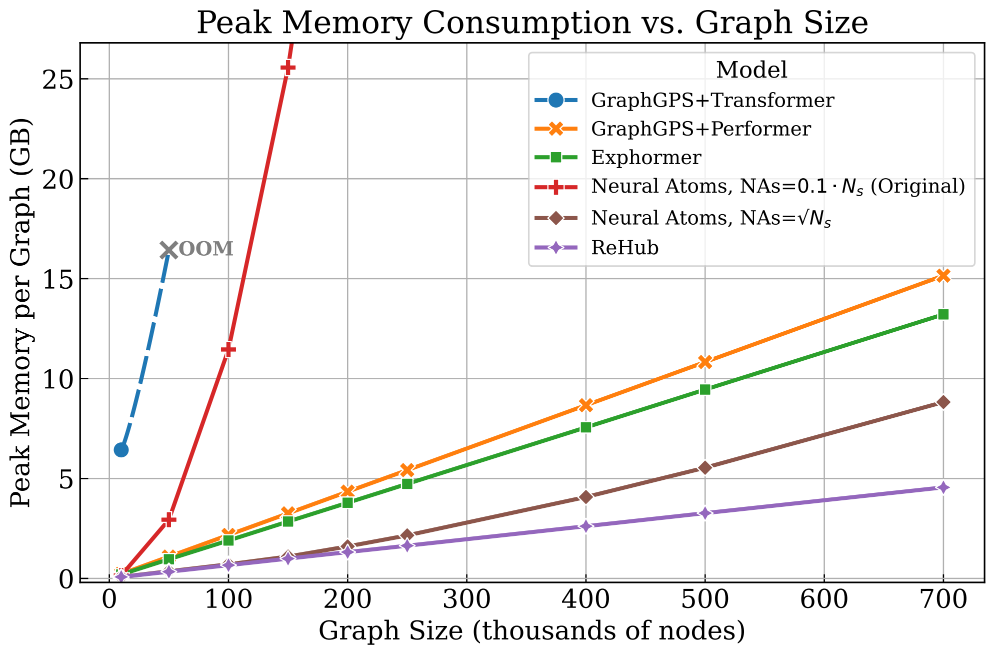

Figure 2: **Peak memory consumption for different architectures**. We compare ReHub to Neural Atoms (Li et al., 2024) and other architectures, and show that ReHub memory consumption is both linear in the number of nodes and requires less memory.

Table 3: Coauthor Physics (Shchur et al., 2018) and OGBN-Arxiv (Hu et al., 2020) test results show ReHub achieves comparable accuracy to Exphormer with significant reduction in memory consumption.

Model	Coauthor Physics		OGBN-Arxiv	
	Peak Memory (GB) ↓	Accuracy ↑	Peak Memory (GB) ↓	Accuracy ↑
GraphGPS (Transformer)	OOM	-	OOM	-
Exphormer	1.77	**97.16** ± **0.13**	2.83	**72.44** ± **0.28**
ReHub (Ours)	**1.13**	96.89 ± 0.19	**2.45**	71.06 ± 0.40

GraphGPS (with vanila Transformer) in the comparison to highlight that these graph sizes are considered challenging to process.

4.3 Ablations and analyses

Long-range spoke update layer components.

As described in Section 3, our long-range spoke update layer is designed to handle long range communication while maintaining memory efficiency. The ablation provided in Table 4 highlights the contributions of the primary design choices, evaluated on PascalVOC-SP which contains an average of 479.4 nodes per graph. (1) Initialization of hub features from spokes vs. learned hub features as parameters, where the latter is analogous to the initialization process of Neural Atoms. The initialization scheme, outlined in Section 3.3, is compared to a configuration where hub features are learned parameters, i.e, $\boldsymbol{h}_{i_h}^0 = \boldsymbol{P}_{i_h} \in \mathbb{R}^d$. Initializing the hubs from spokes significantly improves performance, while using learned hubs results in reduced performance compared to the GNN baseline. (2) The use of a fixed vs. dynamic number of hubs. For the fixed case, we set the number of hubs to 22, which is approximately the square root of the average number of nodes, $\sqrt{479.4} \approx 21.9$. This is compared to setting the number of hubs as $N_h = \sqrt{N_s}$ dynamically, according to graph size. (3) Reassigning spokes to hubs at each layer vs. keeping the same connections fixed across the

Table 4: **Ablation study of long-range spoke update layer components.** We measure the effect of various components of ReHub on top of a GatedGCN MPNN, using the PascalVOC-SP dataset. The number of hubs used per graph (#Hubs): for 22 it is a static amount and for $\sqrt{N_s}$ it is dynamic per graph size. Initial hubs (Hubs Init) can be set as learned parameters or initialized from the assigned spokes as described in 3.3 where we can add a feedforward layer on the spokes (Spokes Enc) before aggregation. Reassignment is as described in 3.4. We use $k = 3$ for all runs.

GNN	#Hubs	Hubs Init	Spokes Enc	Reassignment	PascalVOC-SP (F1 ↑)
+	-	-	-	-	0.3152 ± 0.0045
+	22	Learned (As in NA)	-	-	0.3084 ± 0.0044
+	22	Cluster Mean	-	-	0.3574 ± 0.0065
+	$\sqrt{N_s}$	Cluster Mean	-	-	0.3703 ± 0.0086
+	$\sqrt{N_s}$	Cluster Mean	-	+	0.3797 ± 0.0123
+	$\sqrt{N_s}$	Cluster Mean	+	-	0.3775 ± 0.0040
+	$\sqrt{N_s}$	Cluster Mean	+	+	0.3860 ± 0.0172

Table 5: **Ablation study of spoke-hub assignment strategies.** We analyze the impact of various strategies for hub clustering, initial hub assignment, and hub reassignment on ReHub's performance using the PascalVOC-SP dataset. Accordingly, the results are grouped into three categories, with ReHub's performance reported at the bottom. The clustering methods include random, balanced random, and METIS. For hub assignment and reassignment, we evaluate random, balanced random, feature similarity-based assignment, and no reassignment. Reassignment based on Attention Scores is as described in Section 3.4. We use $k = 3$ for all runs. Our findings show that neither of the random strategies improve the results and actually degrade performance. This further supports our contention that our feature similarity based strategy leverages meaningful spoke-hub relationships.

Clustering	Hubs Assignment	Reassignment	PascalVOC-SP (F1 ↑)
Random	Features Similarity	Attention Scores	0.3503 ± 0.0143
Balanced Random	Features Similarity	Attention Scores	0.3622 ± 0.0060
METIS	Features Similarity	Attention Scores	**0.3860 ± 0.0172**
METIS	Random	Attention Scores	0.3619 ± 0.0083
METIS	Balanced Random	Attention Scores	0.3618 ± 0.0126
METIS	Features Similarity	Attention Scores	**0.3860 ± 0.0172**
METIS	Features Similarity	No Reassignment	0.3775 ± 0.0040
METIS	Features Similarity	Random	0.3514 ± 0.0121
METIS	Features Similarity	Balanced Random	0.3486 ± 0.0106
METIS	Features Similarity	Attention Scores	**0.3860 ± 0.0172**

layers, is shown to improve performance. (4) Including an encoding layer for the spokes prior to aggregation further improves performance.

We provide additional experiments for varying values of hubs ratios and k in Appendix A.4.

Assignment of spokes and hubs.

The clustering into hubs and assignment to spokes are crucial components of ReHub's architecture. The ablation in Table 5 illustrates how different clustering and assignment strategies affect ReHub, by evaluating on PascalVOC-SP. We identify three key constraints that any assignment strategy should satisfy: (1) each spoke must be assigned to exactly k hubs, (2) no hub should appear more than once in a spoke's assignments, and (3) the number of spokes assigned to each hub should be approximately balanced. We therefore devise two baseline strategies to compare against our feature similarity based strategy: Random and Balanced Random. The Random strategy assigns to each spoke k hubs sampled uniformly at random without replacement. While this ensures each spoke has exactly k unique hubs, it does not guarantee a balanced distribution of

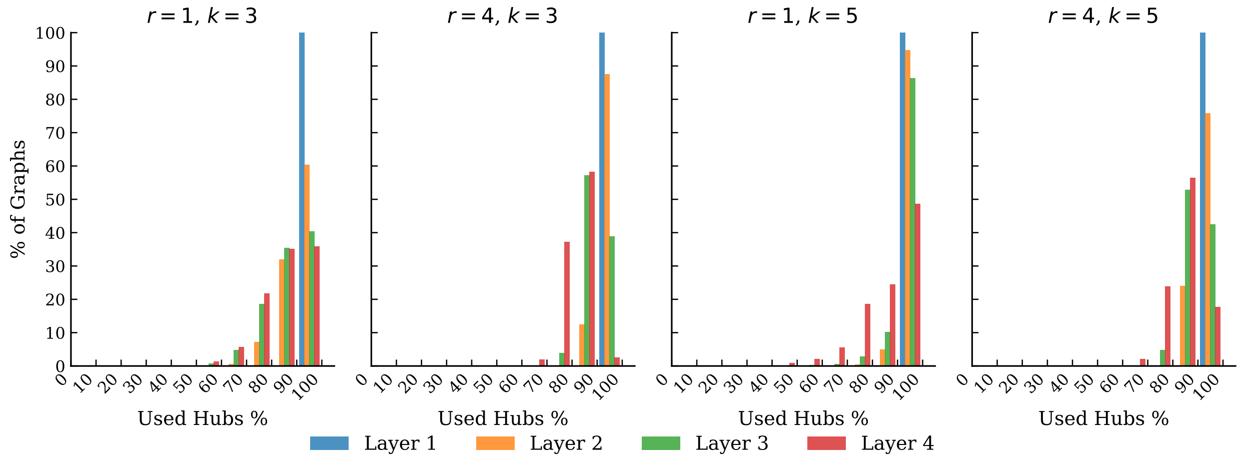

Figure 3: **Hub utilization**. Histogram of used hub percentages across graphs in the PascalVOC-SP validation set, for different configurations of hub ratio r and number of connected hubs k. Each subplot shows hub utilization across four layers. Percentages are grouped using bins of size 10%. Nearly all hubs are utilized across all layers, with the peak typically near 90%-100% usage.

spokes across hubs, and only approaches balance asymptotically for very large graphs. To address this, we propose the Balanced Random strategy, which enforces balance while maintaining some degree of randomness. This assignment task inherently involves a trade-off between computational complexity, randomness, and balance: achieving perfect balance with full randomness is computationally expensive, while a fully random assignment is efficient but unbalanced. The Balanced Random strategy trades off some randomness to achieve approximate balance with linear complexity. Specifically, we begin by randomly permuting the hub indices $0, \dots, N_h - 1$, and for each spoke with index i, assign the k consecutive hubs at positions $(i \cdot u + j) \bmod N_h$ for $j = 0, \dots, k-1$, where u is an integer coprime with N_h. This construction preserves partial randomness, guarantees k unique hubs per spoke, and distributes spokes evenly across hubs. For implementation details, see Appendix A.3. We evaluate both random assignment strategies in three modules: (1) Clustering, as defined in Section 3.3, is used for hub feature aggregation and can be performed randomly or in a balanced random manner, both following the same algorithmic logic described above with $k = 1$. This is compared against METIS, which is the clustering strategy used by ReHub; results are shown in rows 1-3 of Table 5. (2) Initial Hub Assignment is evaluated on random and balanced random strategies, and also compared against ReHub's strategy that assigns hubs based on feature similarity. Results are shown in rows 4-6 of Table 5. (3) Reassignment can be either disabled, performed randomly, balanced randomly, or with the attention score based method as described in Section 3.4. Results are shown in rows 7-10 of Table 5.

Our findings show that both random strategies degrade performance, further supporting the contention that our feature similarity based strategy leverages meaningful spoke-hub relationships.

Hub utilization.

To gain insights into the reassignment dynamics, we measure the level of hub utilization. We define hub utilization U as the number of hubs that have at least one spoke connected to them. Formally, $U = |\{i_h \mid \boldsymbol{E}_{:,i_h} \cdot \mathbf{1}_{N_h} \geq 1\}|$ where $\boldsymbol{E}_{:,i_h}$ represents the set of all spokes connected to hub i_h. The percentage of used hubs is then given by U/N_h, which reflects the proportion of hubs that are connected to at least one spoke. Figure 3 shows histograms of hub utilization across graphs, where the X-axis represents the percentage of used hubs (in bins of size 10), and the Y-axis represents the percentage of graphs falling into each bin. Separate histograms are shown for each layer (Layers 1–4) and for different configurations of hub ratio r and connected hubs k, based on the validation split of the PascalVOC-SP dataset. The results demonstrate that in the vast majority of graphs, nearly all hubs are utilized across all layers, with the peak typically near 90%-100% usage. This finding suggests that the network maintains robust information flow between spokes and hubs, regardless of the number of hubs or the number of spokes per hub. We also present an additional metric based on the Bhattacharyya coefficient in Appendix A.4.

5 Conclusion and future work

In this paper, we introduced ReHub, a novel graph transformer architecture designed to enhance long-range communication in large graphs through dynamic reassignment of virtual nodes. This approach facilitates efficient information passage, enabling effective learning on large graphs while maintaining linear computational complexity and reduced memory usage compared to existing methods. Our experimental results demonstrate that ReHub consistently outperforms Neural Atoms and ranks among the top two methods against various baselines. It achieves competitive accuracy compared to Exphormer with lower memory consumption across all evaluated scenarios. ReHub achieves its efficiency through our proposed reassignment mechanism, which maintains sparse spoke-hub connectivity, as highlighted by our ablation studies. The modular design of ReHub allows for easy integration with various MPNNs.

Future work While consistently improving computational efficiency, our spoke-hub communication maintained strong performance, establishing a foundation for future works. Additionally, our current design lacks inherent support for positional information available in geometric graphs. In future work, we aim to extend our method to support tasks requiring long-range communication on geometric graphs by incorporating positional information into the spoke-hub attention and reassignment mechanisms. We also plan to make the reassignment module learnable, and optimize it towards the prediction task to further boost accuracy. Finally, integrating ReHub into architectures like Exphormer to enable the reassignment of expander graph edges between layers presents a promising direction to further enhance long-range communication.

References

Noga Alon. Eigenvalues and expanders. Combinatorica, 6(2):83–96, 1986.

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. arXiv preprint arXiv:2006.05205, 2020.

Ali Behrouz and Farnoosh Hashemi. Graph mamba: Towards learning on graphs with state space models. In Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp. 119–130, 2024.

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

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? arXiv preprint arXiv:2105.14491, 2021.

Chen Cai, Truong Son Hy, Rose Yu, and Yusu Wang. On the connection between mpnn and graph transformer. In International Conference on Machine Learning, pp. 3408–3430. PMLR, 2023.

Kaidi Cao, Mangpo Phothilimthana, Sami Abu-El-Haija, Dustin Zelle, Yanqi Zhou, Charith Mendis, Jure Leskovec, and Bryan Perozzi. Learning large graph property prediction via graph segment training. Advances in Neural Information Processing Systems, 36, 2024.

Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. Nagphormer: A tokenized graph transformer for node classification in large graphs. arXiv preprint arXiv:2206.04910, 2022.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In International conference on machine learning, pp. 1725–1735. PMLR, 2020.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. arXiv preprint arXiv:2009.14794, 2020.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. Advances in Neural Information Processing Systems, 35:16344–16359, 2022.

Keyu Duan, Zirui Liu, Peihao Wang, Wenqing Zheng, Kaixiong Zhou, Tianlong Chen, Xia Hu, and Zhangyang Wang. A comprehensive study on large-scale graph training: Benchmarking and rethinking. Advances in Neural Information Processing Systems, 35:5376–5389, 2022.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. arXiv preprint arXiv:2012.09699, 2020.

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.

Dongqi Fu, Zhigang Hua, Yan Xie, Jin Fang, Si Zhang, Kaan Sancak, Hao Wu, Andrey Malevich, Jingrui He, and Bo Long. Vcr-graphormer: A mini-batch graph transformer via virtual connections. arXiv preprint arXiv:2403.16030, 2024.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In International conference on machine learning, pp. 1263–1272. PMLR, 2017.

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

Aric A. Hagberg, Daniel A. Schult, Pieter Swart, and JM Hagberg. Exploring network structure, dynamics, and function using networkx. Proceedings of the Python in Science Conference, 2008. URL https://api.semanticscholar.org/CorpusID:16050699.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. arXiv preprint arXiv:1905.12265, 2019.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. Advances in neural information processing systems, 33:22118–22133, 2020.

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.

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on scientific Computing, 20(1):359–392, 1998.

Jeong Han Kim and Van H Vu. Generating random regular graphs. In Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, pp. 213–222, 2003.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.

Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. Advances in Neural Information Processing Systems, 34: 21618–21629, 2021.

Xuan Li, Zhanke Zhou, Jiangchao Yao, Yu Rong, Lu Zhang, and Bo Han. Neural atoms: Propagating long-range interaction in molecular graphs through efficient communication channel. In The Twelfth International Conference on Learning Representations, 2024.

Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. Attending to graph transformers. arXiv preprint arXiv:2302.04181, 2023.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. Advances in Neural Information Processing Systems, 35:14501–14515, 2022.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. arXiv preprint arXiv:1811.05868, 2018.

Ahsan Shehzad, Feng Xia, Shagufta Abid, Ciyuan Peng, Shuo Yu, Dongyu Zhang, and Karin Verspoor. Graph transformers: A survey. arXiv preprint arXiv:2407.09777, 2024.

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In International Conference on Machine Learning, pp. 31613–31632. PMLR, 2023.

Angelika Steger and Nicholas C Wormald. Generating random regular graphs quickly. Combinatorics, Probability and Computing, 8(4):377–396, 1999.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. arXiv preprint arXiv:2111.14522, 2021.

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 Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. stat, 1050(20):10–48550, 2017.

Chloe Wang, Oleksii Tsepa, Jun Ma, and Bo Wang. Graph-mamba: Towards long-range graph sequence modeling with selective state spaces. arXiv preprint arXiv:2402.00789, 2024.

Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. Advances in Neural Information Processing Systems, 35: 27387–27401, 2022.

Qitian Wu, Wentao Zhao, Chenxiao Yang, Hengrui Zhang, Fan Nie, Haitian Jiang, Yatao Bian, and Junchi Yan. Simplifying and empowering transformers for large-graph representations. Advances in Neural Information Processing Systems, 36, 2024.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems, 32(1):4–24, 2020.

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

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? Advances in neural information processing systems, 34:28877–28888, 2021.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. Advances in neural information processing systems, 33:17283–17297, 2020.

Jianan Zhao, Chaozhuo Li, Qianlong Wen, Yiqi Wang, Yuming Liu, Hao Sun, Xing Xie, and Yanfang Ye. Gophormer: Ego-graph transformer for node classification. arXiv preprint arXiv:2110.13094, 2021.

A Appendix

A.1 Datasets

In Tables 6 and 7 we summaries the details of the datasets used for evaluation.

Table 6: Statistics of the five dataset proposed in the long-range graph benchmark. Source: LRGB (Dwivedi et al., 2022).

Dataset	**Total Graphs**	**Total Nodes**	**Avg Nodes**	**Mean Deg.**	**Total Edges**	**Avg Edges**	**Avg Short.Path.**	**Avg Diameter**
PascalVOC-SP	11,355	5,443,545	479.40	5.65	30,777,444	2,710.48	10.74±0.51	27.62±2.13
COCO-SP	123,286	58,793,216	476.88	5.65	332,091,902	2,693.67	10.66±0.55	27.39±2.14
PCQM-Contact	529,434	15,955,687	30.14	2.03	32,341,644	61.09	4.63±0.63	9.86±1.79
Peptides-func	15,535	2,344,859	150.94	2.04	4,773,974	307.30	20.89±9.79	56.99±28.72
Peptides-struct	15,535	2,344,859	150.94	2.04	4,773,974	307.30	20.89±9.79	56.99±28.72

Table 7: Dataset statistics of LRGB, OGBN-Arxiv and Coauthor Physics. Source: Exphormer (Shirzad et al., 2023)

Dataset	**Graphs**	**Avg. nodes**	**Avg. edges**	**Prediction Level**	**No. Classes**	**Metric**
PascalVOC-SP	11,355	479.4	2,710.5	inductive node	21	F1
COCO-SP	123,286	476.9	2,693.7	inductive node	81	F1
PCQM-Contact	529,434	30.1	61.0	inductive link	(link ranking)	MRR
Peptides-func	15,535	150.9	307.3	graph	10	Average Precision
Peptides-struct	15,535	150.9	307.3	graph	11 (regression)	Mean Absolute Error
OGBN-Arxiv	1	169,343	1,166,243	node	40	Accuracy
Coauthor Physics	1	34493	247962	node	5	Accuracy

A.2 Hyperparameters

Long-range graph benchmark. In the experiments done on the datasets: Peptides-func, Peptides-struct and PCQM-Contact we follow the hyperparameters of Neural Atoms (Li et al., 2024). For other datasets we follow the hyperparameters of Exphormer (Shirzad et al., 2023). Although we follow the hyperparameters configurations, there may be subtle changes due to the difference in architecture.

In Tables 8- 11 we provide the hyperparameters used in our experiments.

Table 8: Hyperparameters for the LRGB datasets used for evaluation. For Peptides-func, Peptides-struct and PCQM-Contact some of the hyperparameters are model-specific and presented in additional tables.

Hyperparameter	**PCQM-Contact**	**Peptides-func**	**Peptides-struct**	**PascalVOC-SP**
Dropout	0	0.12	0.2	0.15
Attention dropout	0.2	0.2	0.2	0.2
Positional Encoding	LapPE-10	LapPE-10	LapPE-10	LapPE-10
PE Dim	16	16	20	16
PE Layers	2	2	2	2
PE Encoder	DeepSet	DeepSet	DeepSet	DeepSet
Batch size	256	128	128	32
Learning Rate	0.0003	0.0003	0.0003	0.0005
Weight Decay	0	0	0	0
Warmup Epochs	10	10	10	10
Optimizer	AdamW	AdamW	AdamW	AdamW
# Epochs	200	200	200	300
MPNN	-	-	-	GatedGCN
# Layers	-	-	-	4
Hidden Dim	-	-	-	96
# Heads	-	-	-	8
Hubs Ratio	-	-	-	1
k	-	-	-	3
# Params	-	-	-	590,051

Table 9: Model-specific hyperparameters for PCQM-Contact, and the number of model parameters.

Hyperparameter	**# Layers**	**Hidden Dim**	**# Heads**	**Hubs Ratio**	**k**	**# Params**
GCN	5	300	1	1	3	5,127,540
GCNII	5	100	2	1	3	587,240
GINE	5	100	1	1	3	638,240
GatedGCN	8	72	1	1	3	655,992

Table 10: Model-specific hyperparameters for Peptides-func, and the number of model parameters.

Hyperparameter	**# Layers**	**Hidden Dim**	**# Heads**	**Hubs Ratio**	**k**	**# Params**
GCN	5	155	1	0.5	3	1,372,563
GCNII	5	88	1	0.5	3	452,204
GINE	5	88	2	0.5	3	491,804
GatedGCN	5	88	1	0.5	3	611,044

Table 11: Model-specific hyperparameters for Peptides-struct, and the number of model parameters.

Hyperparameter	**# Layers**	**Hidden Dim**	**# Heads**	**Hubs Ratio**	**k**	**# Params**
GCN	5	155	1	0.5	3	1,376,382
GCNII	5	88	1	0.5	3	453,152
GINE	5	88	2	0.5	3	492,752
GatedGCN	5	88	1	0.5	3	611,992

Large random regular graph. In Table 12 we provide the hyperparameters used in our experiments. The same configuration of hyperparameters is used for all experiments except for model-specific parameters.

Note that although Exphormer does not utilize its expander graph algorithm here the “Add edge index” hyperparameter is enabled and sets the graph edges as the expander edges. Due to the regularity of the graph, *i.e.* having a degree of $d = 3$ for all nodes, the same linear complexity is imposed.

Table 12: Hyperparameters for the forward pass of the large random regular graph dataset for all the models, including the model-specific configuration.

Hyperparameter	**Large Random Regular Graph**	**Exphormer**	**ReHub**
MPNN	GCN	-	-
# Layers	3	-	-
Hidden Dim	52	-	-
# Heads	4	-	-
Add edge index	-	True	-
Num Virtual Nodes	-	4	-
Hubs Ratio	-	-	1
k	-	-	3

OGBN-Arxiv and Coauthor Physics. For OGBN-Arxiv and Coauthor Physics we follow the hyperparameters of Exphormer (Shirzad et al., 2023), adding our configuration of hubs ratio and k. In Table 13 we provide the hyperparameters used in our experiments.

Table 13: Hyperparameters for OGBN-Arxiv and Coauthor Physics datasets used for evaluation, and the number of model parameters.

Hyperparameter	**OGBN-Arxiv**	**Physics**
Dropout	0.3	0.4
Attention dropout	0.2	0.8
Learning Rate	0.01	0.001
Weight Decay	0.001	0.001
Warmup Epochs	5	5
Optimizer	AdamW	AdamW
# Epochs	600	70
MPNN	GCN	GCN
# Layers	3	4
Hidden Dim	80	72
# Heads	2	4
Hubs Ratio	1	1
k	3	3
# Params	240,624	850,281

A.3 Implementation details

In the following we describe in more detail how the architecture is implemented.

We use the open-source code provided by GraphGPS (Rampášek et al., 2022) and available on https://github.com/rampasek/GraphGPS. We merge into that code parts from Exphormer (Shirzad et al., 2023) which are relevant for the training and evaluation of the OGBN-Arxiv and Coauthor Physics datasets.

Clustering. During preprocessing we use the METIS partitioning algorithm (Karypis & Kumar, 1998) to divide each graph to clusters according to the required number of hubs. In practice, we use the python wrapper PyMetis[3] which allows us to map each spoke to a single hub. According to the METIS paper (Karypis & Kumar, 1998), this clustering step runs in approximately $O(N + M)$ time and requires only $O(N)$ additional memory beyond the input graph, for a graph with N nodes and M edges. For graphs where $M \approx O(N)$, as in many practical cases, both the runtime and the additional memory simplify to $O(N)$. The clustering is implemented as a modular preprocessing stage and can, in principle, be replaced with any other clustering algorithm without affecting the overall ReHub framework.

Hub features. Throughout the implementation we use sparse data structures that allows us to keep the complexity linear even when running on multiple graphs simultaneously (*i.e.* in a batch). The hubs are stored in a similar fashion to how spokes are stored in a batch. *i.e.* the features are kept in a 2D matrix $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ together with a 1D graph index matrix $\boldsymbol{B} \in \{0, \ldots, \#\text{Graphs}\}^N$ indicating which graph each node belongs to, where N is the number of nodes in the whole batch.

When aggregating the spokes to calculate hub features we use the `scatter` functions which allows us to aggregate the spokes to two matrices. (1) a 2D matrix $\boldsymbol{H} \in \mathbb{R}^{H \times d}$ and (2) a 1D graph index matrix $\boldsymbol{B}_H \in \{0, \ldots, \#\text{Graphs}\}^H$ indicating which graph each hub belongs to, where H is the number of hubs in the whole batch.

Hub assignment and reassignment. Note that the initial hub assignment is implemented identically to the reassignment algorithm but with i_h^* set to the initial hub found in the clustering step.

Attention. Opposed to other transformer architectures which require the conversion of the spokes to a dense representation, we can keep both spokes and hubs in their sparse representation. For each attention module we use `GATv2Conv` implementation provided in pytorch geometric which accept as input this type of sparse representation. Moreover, this implementation accept as input a bipartite graphs and here we use it to pass information between the graph of spokes and graph of hubs.

Large random regular graph. As mention in Section 4.2, we would like to construct graphs of arbitrary size. To achieve this, we generate a dataset of large random regular graphs, which can be produced at any scale while ensuring connectivity between nodes (*i.e.* , avoiding isolated subgraphs) due to the regularity property, *i.e.* A d-regular graph is a graph where each node has d number of neighbors. To do that we use the `random_regular_graph` (Kim & Vu, 2003; Steger & Wormald, 1999) function from the open-source `NetworkX` library (Hagberg et al., 2008), which takes as an input the number of nodes to be constructed and the degree of each node d. Additionally, to ensure compatibility with the models, we assign random values to the graph's edge attributes, node features, and prediction labels.

Peak memory usage. To sample the peak memory usage of the models we use the function `torch.cuda.max_memory_allocated`. This function returns the peak memory allocation since running the `torch.cuda.reset_peak_memory_stats` function, which we call just before the call to the model.

[3] https://github.com/inducer/pymetis

Balanced random assignment. In Section 4.3 and Table 5, we present an ablation study of the spoke–hub assignment components. Specifically, we discuss a balanced random assignment strategy that balances the number of spokes assigned to each hub while keeping the number of hubs assigned to each spoke strictly equal to k. This strategy ensures per-spoke uniqueness and global balance, at the cost of introducing some structure into the randomness. Given N_h hubs and N_s spokes, we first sample a uniform random permutation π of the hub indices $\{0, \dots, N_h{-}1\}$. We then choose a random stride $u \in \{1, \dots, N_h{-}1\}$ such that $\gcd(u, N_h) = 1$, ensuring a full cycle through the hubs. For each spoke $i \in \{0, \dots, N_s{-}1\}$, we assign it k distinct hubs with indices

$$E_{i,j} = \pi[(i \cdot u + j) \bmod N_h], \quad j = 0, \dots, k{-}1.$$

By construction, each spoke is assigned exactly k distinct hubs, and each hub is assigned to approximately the same number of spokes. The partial randomness of π and u ensures diversity of assignments, while the cyclic structure maintains balanced and distinct connections per spoke.

Algorithm 2 Balanced Random Assignment of Hubs

```
Require: Number of hubs $N_h$, number of spokes $N_s$, number of hubs per spoke $k$
  Sample a random permutation $\pi$ of $\{0, 1, \dots, N_h{-}1\}$
  Sample a random stride $u \in \{1, \dots, N_h{-}1\}$ such that $\gcd(u, N_h) = 1$
  for $i = 0$ to $N_s - 1$ do
    for $j = 0$ to $k - 1$ do
      $E_{i,j} = \pi[(i \cdot u + j) \bmod N_h]$
    end for
  end for
  return Assignment matrix $\boldsymbol{E}$
```

Why is it important for u to be coprime with N_h? If u and N_h are coprime, then the sequence of values $(i \cdot u) \bmod N$ for $i = 0, 1, \dots, N - 1$ forms a permutation of $\{0, 1, \dots, N - 1\}$. That is, each value in the sequence appears exactly once before the sequence repeats. To see this, suppose $(i \cdot u) \bmod N = (j \cdot u) \bmod N$ for some $0 \leq i < j < N$. Then $(j - i) \cdot u$ is a multiple of N, but since u and N are coprime, $j - i$ must be a multiple of N. However, $0 < j - i < N$, a contradiction. Hence all N values are distinct, and since they all lie in $\{0, \dots, N - 1\}$, they must form a permutation of this set. This property ensures that all hubs are used evenly.

A.4 Additional ablations

Long-range spoke update layer components. In addition to the result shown for PascalVOC-SP presented in Section 4.3, we provide in Table 14 the same components analysis for Peptides-func.

Table 14: **Ablation study.** We measure the effect of various components of ReHub on top of a GatedGCN MPNN, using the Peptides-func dataset. The number of hubs used per graph (#Hubs): for 22 it is a static amount and for $\sqrt{N_s}$ it is dynamic per graph size. Initial hubs (Hubs Init) can be set as learned parameters or initialized from the assigned spokes as described in 3.3 where we can add a feedforward layer on the spokes (Spokes Enc) before aggregation. Reassignment is as described in 3.4. We use $k = 3$ for all runs.

GNN	#Hubs	Hubs Init	Spokes Enc	Reassignment	Peptides-func (AP ↑)
+	-	-	-	-	0.5864 ± 0.0077
+	12	Learned (As in Neural Atoms)	-	-	0.5738 ± 0.0027
+	12	Cluster Mean	-	-	0.6626 ± 0.0068
+	$\sqrt{N_s}$	Cluster Mean	-	-	0.6616 ± 0.0063
+	$\sqrt{N_s}$	Cluster Mean	-	+	0.6661 ± 0.0062
+	$\sqrt{N_s}$	Cluster Mean	+	-	0.6612 ± 0.0068
+	$\sqrt{N_s}$	Cluster Mean	+	+	0.6683 ± 0.0069

Sensitivity to hubs ratio For ReHub, a practical guideline for selecting the hubs ratio and k is $r = 1$ and $k = 3$. Figure 4 presents an ablation study on these parameters for the Peptides-func and PascalVOC-SP datasets.

For Peptides-func, the best results are achieved with a hubs ratio of 0.5 for both $k = 3$ and $k = 5$, which may be attributed to the small graph sizes and the potential tendency to overfit on such datasets. On average, each graph contains approximately 150 nodes, resulting in roughly 6 hubs. Furthermore, it is notable that for other hubs ratio values, the results remain consistent, with an average performance between $AP = 0.66$ and $AP = 0.67$, indicating the robustness of our method.

For PascalVOC-SP, which includes larger graphs with an average of approximately 500 nodes per graph, a different trend is observed compared to Peptides-func. Specifically, for varying values of k, the optimal performance is achieved with different hubs ratios. However, the performance variation outside the optimal hubs ratio is relatively minor, with the best results obtained when $r = 1$ and $k = 3$.

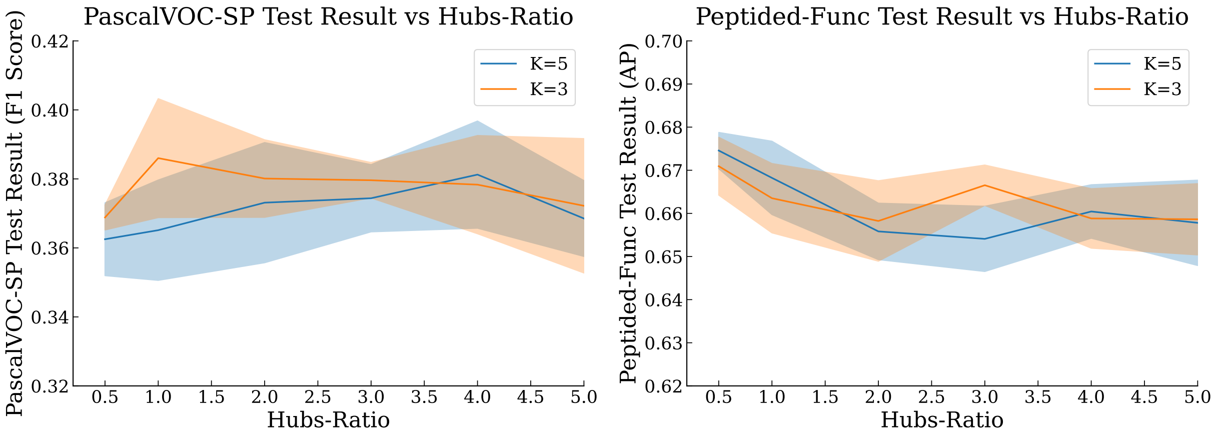

Figure 4: Results for various hubs ratio and k, which is the number of hubs each spoke is connected to. We shown this on PascalVOC-SP (Left) and Peptides-func (Right) datasets with $k = [3, 5]$ and $r = [0.5, 1, 2, 3, 4, 5]$.

Bhattacharyya Coefficient vs. Uniform Distribution The Bhattacharyya Coefficient for a discrete probability distributions P and Q is measurement of how similar the two probability distributions are. It is defined as:

$$\mathrm{BC}(P, Q) = \sum_{x \in \mathcal{X}} \sqrt{P(x)Q(x)},$$

$BC(P, Q)$ lies between 0 and 1, where the higher the coefficient the more similar the distributions are. In our setup, P is the distribution over spokes connection per hub; *i.e.* it is the number of spokes connected to each hub, divided by the overall number of connections from spokes to hubs (so that the result is indeed a probability distribution). We then set Q to be the uniform distribution, *i.e.* $1/N_h$ for each hub. By doing so, we can see how close the actual distribution P is to the uniform distribution, where uniform indicates the optimal balanced assignment of spokes to hub – where every hub has exactly the same number of spokes. For convenience, we define the Bhattacharyya Percentage as the Bhattacharyya Coefficient multiplied by 100.

In Figure 5 we present a graph illustrating the percentage of graphs with a Bhattacharyya Percentage below a given threshold for the PascalVOC-SP dataset. As in Section 4.3 this is shown for each layer, and for varying values of hubs ratios and k on the validation split of the dataset. The results demonstrate that regardless of the number of hub and number of hubs per spoke, most graphs have a Bhattacharyya Percentage above 80%. This suggests that our reassignment method spreads nodes quite evenly across the various hubs, and does not create high concentration of spokes which remain connected to only few hubs.

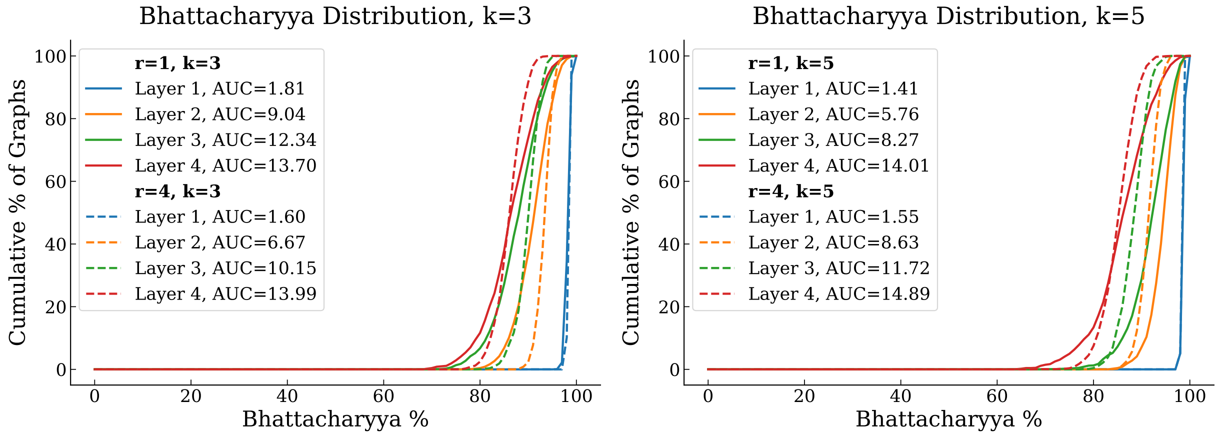

Figure 5: Percentage of graphs with a Bhattacharyya Percentage below a given threshold for the validation split of the PascalVOC-SP dataset. Results are shown for varying k and hubs ratio r. Left: $k = 3$ with $r \in \{1, 4\}$. Right: $k = 5$ with $r \in \{1, 4\}$.