Overlay Neural Networks for Heterophilous Graphs

Giuseppe Pirrò\textsuperscript{a;*}

\textsuperscript{a}Department of Mathematics and Computer Science, University of Calabria, Rende (CS), Italy
ORCiD ID: Giuseppe Pirrò https://orcid.org/0000-0002-7499-5798

\textbf{Abstract.} Graph Neural Networks (GNNs) have become increasingly popular for their ability to capture complex relationships within graphs by aggregating node neighbor information. However, in graphs exhibiting high levels of heterophily relevant within graphs by aggregating node neighbor information. To tackle the problem of incorporating long-range relevant neighbors into the GNN node aggregation mechanism, this paper introduces the Overlay Graph Neural Networks (OGN) model. OGN is inspired by P2P overlay networks, where the idea is to find neighbor peers (nodes) that, although not directly connected to a given node (a peer), are semantically similar and could favorably improve both query routing and query results. In our context, the network is the graph, and the routing is the message passing a GNN performs to aggregate node features. OGN networks are built by stacking one or more overlay layers, each taking as input the graph and a node feature matrix either available or derivable (e.g., by analyzing the graph’s structure). Each overlay layer combines base embeddings, learned by considering node features and short-range node neighbors, with overlay embeddings computed by projecting nodes with similar features close in an overlay space and then aggregating (overlay) neighbor nodes via a sliding window attention mechanism. Base and overlay embeddings are combined to capture nodes’ immediate and global context in a graph. We evaluate OGN in a node classification task using state-of-the-art benchmarks and show that OGN is competitive with the advantage of being easily portable to any existing GNN model.

1 Introduction

The massive amounts of information available in the form of graphs fostered a variety of machine/deep learning techniques centered on graphs [34, 7]. Graph Neural Networks (GNNs) is a prominent graph embedding approach that supports downstream applications like link prediction and node classification [30]. One major challenge in GNNs is the problem of capturing complex relationships between nodes that are not immediately adjacent to each other, especially in heterophilous graphs [35]. This is a critical challenge in GNNs, as their inability to capture such long-range connections can significantly impact their performance [35]. One way to solve the problem is to stack different GNN layers, as each layer performs local aggregation over the immediate neighbors. However, establishing the correct number of layers is complex; a too-low value can miss important neighbors, while a too-high value can lead to over-smoothing [10]. Not to mention the increasing computational demand of training deep GNNs [15]. Therefore, to learn from graphs with heterophily, enhancing node neighborhoods with long-range (relevant) nodes remains an open challenge.

\textbf{Related Work.} There are three main strands of research: (i) approaches that modify the graph; (ii) approaches that change the GNN learning architecture; (iii) approaches that discover neighbors. Zheng et al. [35] provide a comprehensive survey. As for (i), the idea is to insert edges between (distant) similar nodes (e.g., [2]) or rely on surrogate computational graphs (e.g., [25]) in which different types of edges indicate different neighbors. As for (ii), the idea is to change the GNN learning architecture by relying on label-wise [6] or attribute-wise [32] message passing or modify the aggregation and combination mechanisms (e.g., H2GCN [37]). As for (iii), approaches like Geom-GCN [20] map the graph to the 2D space and leverage a distance metric to discover neighbors. Other approaches (e.g., GPN [33], NLGNN [17]) adopt variants of the attention mechanism. Yet other approaches are based on (too) specialized models (e.g., [18, 37]). We discuss a variety of competitors in Section 5.2.

\textbf{OGN Overview.} Unlike the previous works, we address the problem of node classification in graphs with heterophily by combining local and long-range relevant neighbors in the Overlay Graph Neural Networks (OGN) model. The key idea is to enable a virtual overlay graph where a 1D node ordering is learned from the data, with the intuition that nodes that are close in the ordering are neighbors in the overlay graph. Then, overlay neighbors are combined with local neighbors to obtain the final node representations. This avoids creating surrogate graphs or changing the message-passing scheme. OGN networks are built by stacking overlay layers, each taking as input the graph $G$ and a node feature matrix $X$ either available (e.g., initial node features) or derivable by analyzing the graph (e.g., ordered node neighbor degrees treated as additional node features) and thus amenable to different instantiations (see Fig. 1 (a)). The layer learns and combines base embeddings and overlay embeddings. Base embeddings are learned by leveraging node features and combining information from immediate node neighbors (a few hops away). The challenge now involves identifying and incorporating relevant long-range neighbors. OGN resorts to embedding similarity as an indicator of mutual node relevance; to do so, embeddings learned from the initial node features and neighbors are projected into 1D scores and sorted so that the positions of (node) embeddings reflect node similarity (see Fig. 1 (b)). As base embeddings are learned, the intuition is that nodes having similar features (and neighborhoods) will be projected closer than nodes having dissimilar features and neighbors in the 1D embeddings space. At this point, overlay embeddings are obtained using a sliding-window attention approach [13].
point is that instead of computing attention scores of each node towards each other node as typically done by attention mechanisms [26] or by approaches that compute pairwise node similarity to establish node closeness (e.g., [25]), only nodes within a window of size \( w \) (\( w/2 \) on the left and \( w/2 \) positions on the right of the node) are considered. The intuition is that in a graph with \( N \) nodes, since nodes are already ordered in the overlay space, a sliding window of size \( w \) can capture the most relevant nodes. For example, in Fig. 1 (c), to compute the overlay embedding of \( v_1 \) with a window size \( w = 2 \), information from its overlay neighbors \( v_3 \) and \( v_2 \) is aggregated. Interestingly, the ordering in Fig. 1 (b) induces an overlay graph, which is different from the original graph in general. In the overlay graph in Fig. 1 (c) built considering \( w = 2 \), we note that \( v_1 \) is a neighbor of \( v_5 \), which is not the case in the original graph in Fig. 1 (a). We also note that the overlay graph is implicit and does not need to be materialized. Another interesting feature of OGN is that one can stack several overlay layers, each considering a different node feature matrix or configuration (e.g., window size), and build an end-to-end learning framework.

Contributions. We introduce the Overlay Graph Neural Network (OGN) model to capture long-range neighbors in heterophilic graphs effectively. We got inspiration from Peer-to-Peer Networks (P2P) [5], where an overlay was a virtual network on top of the original network topology, and Longformer architecture [13] that addresses the problem of avoiding pairwise attention computation for sentences. We contribute:

1. A mechanism that reduces the problem of finding long-range neighbors to that of learning a node ordering starting from base embeddings. The advantage is that this approach does not need to add edges to the original graph, constructing surrogate graphs, or changing the GNN message passing scheme.
2. A novel overlay embedding mechanism aggregates ordered embedding resulting from (1) using a sliding window attention mechanism, which instead of computing pairwise attention scores, limits the number of comparisons using a sliding window.
3. OGN networks, an integrated and flexible learning framework that combines different overlay layers to generate node embeddings for downstream applications. We discuss node classification.
4. A comprehensive and evaluation of the state-of-the-art based on up-to-date novel baselines and datasets [22].

We evaluated OGN on node classification benchmarks on heterophilic graphs and show that they achieve state-of-the-art results featuring a simple way to incorporate (different kinds of) long-range neighbors via the overlay layer abstraction.

Outline. The remainder of the paper is structured as follows. We provide some background in Section 2. We introduce the building block of OGN in Section 3. In Section 4, we show how to use OGN for node classification. We report on an experimental evaluation in Section 5. We conclude and sketch future work in Section 6.

2 Background and Problem Statement

Let \( G = (V_G, E_G) \) be an undirected graph with \( N = |V_G| \) nodes, \( V_G = \{v_1, v_2, \ldots, v_N\} \) is the set of nodes and \( E_G = \{e_{ij} \mid v_i, v_j \in V_G\} \) the set of edges. The adjacency matrix \( A \in \mathbb{R}^{N \times N} \) is defined by \( A_{ij} = 1 \) if \( e_{ij} \in E_G \) and 0 otherwise. We are interested in graphs endowed with node features. Node features are represented by a set of \( M \) matrices, where each matrix is defined as \( \mathbf{X}_i^m = \mathbf{X} \times F \) for \( m \in [1, M] \), where \( \mathbf{X}_i^m \) is the \( F \)-dimensional feature vector of node \( v_i \) in the matrix \( m \). Examples of feature matrices are, for instance, one-hot encoding of keywords or attributes for each node. We denote by \( \mathcal{N}(u) \) the neighbors of \( u \) that are \( k \)-hops away. We denote by \( \mathcal{N}^\leq_k(u) \) the set of overlay neighbors of \( u \) in the \( q - th \) overlay layer.

2.1 Graph Neural Networks

We now introduce Graph Neural Networks (GNN). Let \( H^{(0)} \in \mathbb{R}^{N \times D_0} \) be the initial node feature matrix and \( h_i^{(0)} \) the \( D_0 \)-dimensional feature vector of node \( v_i \). A GNN is based on the idea of stacking convolutional layers to perform node feature transformation and aggregation of neighbor node features [10]. The updating process of the \( l \)-th GNN layer of each node \( v_i \in V_G \) is:

\[
h_i^{(l)} = \text{UPD} \left( h_i^{(l-1)} \right) \quad \text{AGGR} \left( \{h_j^{(l-1)} : j \in \mathcal{N}(i)\} \right)
\]

(1)

where \( h_i^{(l)} \) is the representation of node \( v_i \) at layer \( l \), and the neighbors of \( u \) in \( \mathcal{N}(u) \) do not include \( u \). The AGGR function aggregates representations (e.g., via LSTM, MaxPooling) only from the direct neighbors generating a message, while UPD (e.g., linear combination) combines the previous representation of the node with the message from the neighbors.

2.2 Problem Statement and Challenges

In the classical GNN setting, the embedding \( h_i \) of a node \( v_i \) is computed by combining and aggregating information from \( v_i \)’s neighbors and \( v_i \)’s itself. In general, information propagated from 1-hop neighbor nodes tends to be (too) similar to that of the current node, which may undermine the performance of a GNN. Hence, to capture long-range neighbors, one way is to stack different GNN layers. There are two fundamental problems with this scenario. On the one hand, it is now well-understood that GNNs suffer from over-smoothing (see, e.g., [19]); the receptive field of a node tends to include more and more noise as soon as the number of layers increases. On the other
hand, a small number of layers can miss important long-range neighbors. Hence, the issue of incorporating relevant long-range nodes while avoiding noise becomes crucial. This paper aims to tackle this critical issue by introducing OGN and focusing on the problem of node classification. The problem that we face in this paper can be stated as follows:

**Problem 1** Given a graph \( G = (V_G, E_G) \) with a small set of labels \( Y_L \) for the node set \( V_G \), we aim to devise a learning architecture based on OGN, which can accurately predict the labels of the unlabeled nodes, i.e., \( f_O(G, Y_L) \rightarrow \hat{Y}_U \) by incorporating long-range relevant neighbors. Here, \( f_O \) is the function we aim to learn, and \( \hat{Y}_U \) is the set of predicted labels for unlabeled nodes.

To solve Problem 1, we identify the following three main challenges:  
- **C1**: Learning Overlay Neighbours;  
- **C2**: Aggregating Overlay Neighbours;  
- **C3**: Devising an end-to-end framework for node classification.  

We tackle challenges **C1** and **C2** in Sec. 3 and challenge **C3** in Sec 4.

3 Overlay Layer

In this section, we introduce the building block of OGN-based architectures, the overlay layer whose architecture is outlined in Fig. 2. An overlay layer takes the graph and a feature matrix as input and produces node embeddings. Due to its generality, the model can consider different node feature matrices, some available while others can be derived by looking, for instance, at specific node properties. For example, we will discuss in the experimental evaluation how available node features (e.g., one-hot encoding of words) can be combined with structural node features (e.g., ordered node neighbor degree sequences) to capture different similarity perspectives. An overlay layer is composed of three main modules: a *base embedding module*, an *overlay embedding module*, and a combination module. The generality of the definition leaves room for different instantiations of these three components, as we will discuss.

3.1 Base Embedding Module

The base embedding module takes the graph and a feature matrix as input and produces base embeddings. As shown in Fig. 2, it can be instantiated with a simple Multi-Layer Perceptron or a more sophisticated architecture like GCN [14], GAT [27] or GraphSAGE [11]. We generally indicate with \( E_B : \mathbb{R}^k \rightarrow \mathbb{R}^d \) a base embedding module that learns \( d \)-dimensional vectors, one for each node in the graph, starting from \( k \)-dimensional node feature vectors. Let \( G=(V_G, E_G) \) be the graph and \( \mathcal{X} \) be the feature matrix input of the overlay layer. We have that: \( h_i = E_B(x_i) \) \( \forall v_i \in V_G, x_i \in \mathcal{X} \), where \( h_i \) is the embedding of node \( v_i \) and \( x_i \) its feature vector. We observe that when instantiation \( E_B \) with a \( L \)-layer GNN-like architecture (e.g., GAT), \( h_i \) can capture the node feature information from nodes within a local neighborhood, and information from nodes \( L \) hops away. However, relevant neighbors that are more than \( L \)-hops away are missing. Not to mention the problem of over-smoothing in GNNs.

3.2 Overlay Embedding Module

We came up with the idea of OGN taking inspiration from Peer to Peer Networks (P2P) [5]. In P2P networks, the idea of semantic overlay networks was to find neighbor nodes that, although not directly connected to a given node (a peer), are semantically similar and could favorably improve query routing and query results [21]. In our context, the network is the graph, and the routing is the message passing a GNN performs to aggregate node features. The peculiarity of our approach is that overlay neighbors are learned starting from base node embeddings. While base embeddings capture the immediate neighborhood of nodes (ideally up to a few hops), overlay embeddings can capture the global context of nodes, including longer-range neighbors. We describe the overlay layer, the building block of OGN, which works as follows (see Fig. 2).

**Relevance score computation**: base embeddings \( H_L \) learned using node features and the immediate neighbors (Section 3.1) are fed as input to an MLP network, which returns a set of relevance scores \( S \), one for each node \( v_i \) in the graph, that is, \( S = MLP(h_i) \). We note that the output dimension of the MLP is 1 (a score for each node) and that the weights in the MLP will be updated during training. These scores resemble attention scores if one considers the set of nodes as the key and value vectors and the weights (of the MLP) as the query vector. This step aims to map nodes with similar features close together.

**Overlay graph induction**: The scores (and the corresponding base node embeddings) are sorted. This step is crucial since similar nodes will be placed close in the ordering. From an abstract perspective, the score ordering induces an overlay graph, that is, a graph where node neighbors (overlay neighbors) are determined by the position in the ordering instead of the existence of a direct edge. We note that this graph is not materialized. This tackles challenge **C1** (Section 2.2).

**Overlay neighbor aggregation**: The crucial question is establishing the neighbors for each node in the induced overlay graph. In other words, for each node, one needs to choose how many other nodes and which nodes should be considered as overlay neighbors and aggregate the corresponding features. For this purpose, the overlay layer leverages the overlay neighbor aggregation component detailed in Fig. 3. In principle, each node can be relevant to each other node; this reasoning resembles the attention mechanism, where the relevance of each word to each other in a sentence is assessed via attention scores [26]. However, in our context, where the "sentence" is the ordered sequence of nodes in a graph, we can avoid pairwise attending nodes because nodes most relevant to each other are assigned similar scores. Therefore, for node aggregation, we adopt a similar reasoning that the Longformer [13], and leverage a sliding window attention mechanism, where given a window size \( w << N \), each node attends \( w/2 \) nodes on its left and \( w/2 \) nodes on its right in the ordered node sequence. For example, node \( v_3 \) in Fig. 3 with
w = 2 attends vi and v2, which are its closest overlay neighbors. The overlay embedding of h5, denoted ad h5 will aggregate information from the embeddings of v1 and v2 appropriately weighted with the attention scores α5,1 and α5,2, respectively. A similar reasoning applies to the other nodes, leading to overlay embeddings. This phase tackles challenge C2 (Section 2.2).

3.3 Combination module

The combination module combines base and overlay embeddings. To let the model separate the contribution of the different kinds of neighbors, we express the final representation of a node vi from an overlay layer as follows: 

\[ h_i^{\text{final}} = \text{UPD} (h_i^{(1)}, h_i^{(2)}, \ldots, h_i^{(L)}) \]  

The above formula is reminiscent of jumping connections [31] with the critical difference that includes the contribution of the overlay. In many applications, UPD can be defined as the identity function, meaning that we concatenate the representations from each layer and the overlay representation. We can also consider other options such as LSTM attention or max-pooling like in GraphSage [11]. In general, concatenation was observed to be helpful for the general context of GNNs [10] without overlay edges. We speculate that this design choice also applies to overlay embeddings learned from different overlay layers as even in this case, the aim is to keep their contribution to \( h_i^{\text{final}} \) separate. The combination is then passed to an MLP with as many outputs as the number of classes.

4 OGN for Node Classification

To tackle challenge C3 (Section 2.2), we introduce the OGN architecture for node classification. The architecture stacks m overlay layers giving a broad range of configuration choices in their definitions; one can vary the type of node feature matrix, the base embedding module, or the sliding window size for overlay aggregation. The architecture of OGN for node classification is shown in Fig. 4. To let the model separate the contribution of the different overlay layers, we express the final representation of a node vi from a set of L overlay layers as follows:

\[ h_i^{\text{final}} = \text{UPD} (h_i^{(1)}, h_i^{(2)}, \ldots, h_i^{(L)}) \]  

5 Experimental Evaluation

We now discuss the experimental evaluation of OGN and the comparison with the state-of-the-art. We start with a description of the datasets and the experimental setting (Section 5.1) and the competitors considered (Section 5.2). Then, we discuss the comparison with the state-of-the-art (Section 5.3) and an ablation study (Section 5.4).

5.1 Datasets

We considered state-of-the-art datasets recently introduced by Platonov et al. [22]. The authors identified several problems in popular benchmarks [35] used to evaluate node classification approaches in graphs with heterophily. It was shown that the performance of many systems significantly degrades when inspecting and cleaning these popular benchmarks. Problems include duplicate nodes in the Squirrel and Chameleon datasets and the very small size (≈200 nodes) and imbalanced class distribution of other datasets (i.e., Texas, Wisconsin, Cornell). Hence, novel, carefully created datasets covering a broader set of domains and being larger have been proposed. These datasets, used in the evaluation, are summarized in Table 1.

The Chameleon and Squirrel datasets we considered are the filtered version constructed where duplicate nodes were removed [22]. Roman-empire is based on the Roman Empire article from English Wikipedia. The graph is a chain graph with additional shortcut edges corresponding to syntactic dependencies between words. The class of a node is its syntactic role (we select the 17 most frequent roles as unique classes and group all the other roles into the 18th class). Amazon ratings is based on the Amazon product co-purchasing network metadata dataset. Nodes are products (books, music CDs, DVDs, VHS video tapes), and edges connect products frequently bought together. Rating values are represented via five classes. Node features are embeddings for words in the product description. Minesweeper is a synthetic dataset inspired by the Minesweeper game where each node (cell) is connected to eight neighboring nodes (except nodes at the edge of the grid, which have fewer neighbors). The node features are one-hot-encoded numbers of neighboring mines. Tolokers is based on data from the Toloka platform. The nodes represent tookers (workers) participating in at least one of 13 projects. An edge connects two tookers if they have worked on the same task. Node features are based on the worker’s profile information and task performance statistics. The Questions dataset is based on data from the question-answering website Yandex. Nodes are users, and an edge connects two nodes if one user answers the other user’s question during a one-year time interval. Node features are the embeddings for words in the user description.

Dataset splits and hyperparameter tuning. In the experiments, we used the dataset splits provided by Platonov et al. [22] and available online1 Specifically, the authors fix 10 random 50%/25%/25% train/validation/test splits. All models’ performances are obtained by computing the averaged results and the standard deviation over the splits. We report accuracy for multiclass classification datasets (roman-empire, amazon-ratings); for binary classification datasets (minesweeper, tolokers, questions), we report ROC AUC. The hyperparameters, including weight decay, dropout, initial learning rate, and patience for learning rate decay, are tuned by searching on the validation performance. The hyper-parameters we tune for all baselines are the number of layers \((1, 2, 3, 4, 5)\) and the hidden size dimensions \((128, 256, 512)\). For OGN, we also considered different sliding-window sizes \((5, 10, 20, 30, 50)\). For all the other hyperparameters, we use the same values across all baseline models and datasets. We implemented our approach using PyTorch2 and DGL3 by incorporating it in the evaluation pipeline4 made available by Platonov et al. [22].

5.2 Approaches considered

We choose several representative neural architectures as our baselines according to the state-of-the-art results on the considered datasets. The baselines include graph-agnostic approaches like

\[1 \text{https://github.com/yandex-research/heterophilous-graphs}\]

\[2 \text{https://pytorch.org}\]

\[3 \text{https://www.dgl.ai}\]
**ResNet** [12], which treats all nodes as independent samples and does not have access to the graph topology. We also consider two variants of **ResNet** that consider graph structural information either (ResNet+SGC) by multiplying the initial node feature matrix with a power of normalized graph adjacency matrix [29] or by augmenting node features with the rows of the adjacency matrix (ResNet+adj).

In terms of GNN architectures, we considered GCN [14] and GraphSAGE [11] (with mean aggregation), GAT [27], which uses a simple attention-based aggregation, Graph Transformer (GT) [24], which is an adaptation of the popular Transformer architecture [26] where nodes having similarity above a threshold $th_s$ that we set to $th_s=0.8$ (this was the best value). The similarity was computed as the cosine between node feature vectors. The goal was to assess whether adding edges upfront performs better than learning overlay embeddings via the overlay embedding module.

Hence we can build OGN models stacking overlay layers. As an example, OGN$^{feat}+OGN_{struc}$ is a two-layer model where in both layers, base embedding is computed using a two-layer GAT. In contrast, overlay embeddings are computed using two different feature matrices (initial node features in the first and structural features in the second). In this case, incorporating two node feature matrices can help to assess whether considering different node similarity perspectives (node feature, structural features) brings some improvement.

Experiments were run on a Linux machine with 16GBs of RAM and a Tesla P100 GPU$^5$. We train each model for 1000 steps and select the best step based on the performance of the validation set. The results are the average of 10 runs (95% c.i).

**5.3 Evaluation results**

The results of the evaluation are shown in Table 2. Performance and standard deviations are reported on a color scale from red (worst/broadest) to green (best/narrowest). The approaches considered are divided into four main groups, which we analyze separately.

1. **Graph-oblivious systems**: These approaches achieve notable performance in the filtered version of the squirrel (SquiF) and chameleon (ChamF) datasets. In particular, ResNet+adj is among the best-performing systems in both datasets. The situation completely changes when looking at the recently introduced datasets. Here these models are consistently outperformed by the approaches that consider the graph structure (group 2) while performing reasonably well compared to group 3, that is, specialized approaches. One example is the RomanEmp, where ResNet+adj is the worst-performing system. Ditto for MineS and (partially) for Tolokers. By looking at the standard deviations of this group

---

$^4$ https://github.com/yandex-research/heterophilous-graphs

$^5$ We thank Amazon for providing the computing infrastructure
of approaches, we note that it is reasonably small in SquiF, RomanEmp, and Questions, while it is broader in ChamF.

2. **GNN-like architectures**: This group includes several variants of GNN models. We note that the performance of this group is reasonable on SquiF (where a bare GCN obtains the second-best result) RomanEmp, where GAT-sep is the best-performing system, while it tends to be worse on ChamF and MineS. In particular, we note that these models perform better than many approaches in group 3, which includes specialized models to learn from graphs with heterophily. This was recently observed by Plataniotis et al. [22]. However, their study did not include NLGNN [17].

3. **Specialized methods**: This group of approaches includes various recently proposed systems specifically designed to learn from graphs with heterophily. As an example, FSGNN, which decouples the node feature aggregation step and depth of graph neural network, was one of the best-performing systems in the classical datasets squirrel, chameleon, etc. However, we note that it fails short in these new datasets, carefully designed to test systems in heterophyous graphs. Ditto for FAGCN and JacobConv.

4. **Long-range neighbors**: The last group includes OGN and our direct competitor NLGNN; both systems leverage the idea of incorporating long-range neighbors. It is interesting to observe that NLGNN performed quite well in the original versions of squirrel and chameleon; we observe that while its performance is also good on SquiF, the situation is different in ChamF where it performs worse than many systems. We note that OGN performs consistently better than other approaches on almost all the datasets. Exceptions are RomanEmp and questions where, surprisingly, it is outperformed by approaches in group 2, that is, variants of simple GNN systems. As compared to approaches in group 3, OGN seems to provide a more consistent quality of results while these approaches show higher variability. For example, FAGCN performs quite well on SquiF and ChamF while not among the top-performing system in any other dataset. By focusing more on our direct competitor NLGNN, we observe that OGN includes different node features (originally available and structural features), allowing for more flexibility in defining (and composing) the overlay layers. The results hint that including different features (available, structural) is the reason for the better performance. We also tested OGN configurations using the overlay layer OGN_{sim}. The configuration OGN_{feat}+OGN_{sim}^{\text{GATS}} was the best performing even though results were inferior to OGN_{feat}+OGN_{struc}^{\text{GATS}} by ~10%. This layer requires a significant upfront effort to compute pairwise similarity score (used to determine overlap neighbors); moreover, it is susceptible to the similarity threshold. In fact, by lowering the threshold from 0.8 to 0.6, performance degrades significantly. Finally, we also tested the three-layer configuration OGN_{feat}+OGN_{struc}^{\text{GATS}}+OGN_{sim}^{\text{GATS}}− which, besides the higher computational demand for training, we noted no increase in the performance. Perhaps, adding a further overlay layer introduces noise instead of favoring the augmentation of neighbors in the aggregation phase.

### 5.3.1 Homophily level and OGN

In this section, we shed light on the benefit of the overlay graph induced by OGN. To do so, we investigate whether the overlay graph induced by the overlay neighbors learned using the sliding window attention mechanism brings some changes in terms of homophily level as compared to the original graph. In particular, for each dataset, we ran our OGN and materialized the induced overlay graph. We considered each node’s overlay neighbors (refer to Fig.1 for an example). Then, we computed the adjusted homophily [22] in the original graph and compared it with the value in the induced overlay graph. Formally, edge homophily is

$$h_{edge} = \frac{|\{(u,v) \in E : y_u = y_v\}|}{|E|}$$.
where \( y_u \) is the label of a node \( u \) and \( E \) is the set of edges. Adjusted homophily is based on the edge homophily and can be computed as follows:
\[
h_{adj} = h_{edge} - \sum_{k=1}^{C} \frac{D_k^2}{(2|E|)^2},
\]
where \( D_k \) is the degree of a node.

The intuition is that the level of adjusted homophily should increase since the goal of OGN is ultimately to make (useful) long-range neighbors direct neighbors. The results of this analysis are shown in Fig. 5.

### 5.4 Ablation study

We performed an ablation study to understand the impact of the elements in the OGN architecture considering. We tested different base embedding modules and configurations with only the OGN\(_{feat}^V \) overlay layer built from the available node features (referred to as L1) and OGN\(_{struct}^V \) the overlay layer built upon structural features (referred to as L2); in both cases \( x \in \{ \text{GCN, GAT, GAT-Sep, GCN-Sep}\} \).

We observe that the configuration with only one overlay layer brings some improvement in almost all datasets as compared to the OGN_{sym}^V baseline, showing that a dynamic way of determining node neighbors is, in general, more viable than the strategy that precompute node similarity and established neighbors using a threshold (0.8 in this case). A significant improvement can be noted when two overlay layers are considered with base embeddings computed using GAT-Sep. This GAT variant keeps ego and neighbor embeddings separate during aggregation. We note that also OGN keeps base and overlay embedding separate during the combination (see Fig. 4). Perhaps this design choice is crucial in a context where one needs to incorporate node neighbors in a more principled way, like in heterophilous graphs.

### 6 Concluding Remarks and Future Work

We studied a novel problem in the GNN landscape: What happens when the notion of the neighbor of a node is not only topological? We proposed an approach called OGN, which builds an overlay graph transparently on top of the original graph where long-range relevant neighbors, which we call overlay neighbors, can be incorporated as neighbors. Each overlay layer can account for different node features; we have shown an architecture for node classification based on two overlay layers that bring performance comparable to the state-of-the-art. We plan to develop a more refined version of our proposal extended to knowledge graphs [9].

---

**References**


[24] Susheel Suresh, Vinith Budde, Jennifer Neville, Pan Li, and Jianzhu Ma, ‘Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns’, Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Min-