# Quarter 1 Project: Benchmarking Graph Learning Models

**Quentin Callahan**
qcallahan@ucsd.edu

**Esther Cho**
ehcho@ucsd.edu

**Yusu Wang**
yusuwang@ucsd.edu

**Gal Mishne**
gmishne@ucsd.edu

**Abstract**

Graph Neural Networks (GNNs) have become a central tool for analyzing graph-structured data, giving advanced performance on various node and graph level tasks. In this report, we compare and explore the performance of various GNN architectures on diverse datasets with distinct task requirements and structure. This paper includes the following models: GCN (graph convolutional network), GIN (graph isomorphism network), GAT (graph attention network) and GPS (a graph transformer model). We investigated node classification on the Cora dataset, graph classification on the Enzyme and IMDB datasets, and graph regression on Peptide-Struct dataset, which involved long-range dependencies. Through many experiments and performance comparisons, we study the strengths and weaknesses of various architectures across datasets, providing insights into their applicability for future developments in graph representation learning. Our findings demonstrate multiple insights, including that traditional GNNs, while effective on localized datasets like Cora, face limitations when handling tasks that require long range reasoning, as seen in Peptides-Struct. Graph Transformers are equipped with attention mechanisms that better capture such long-range dependencies, but are known to be more computationally complex.

Code: `https://github.com/estherch0/Graph-Learning-Models`

# 1 Introduction

## 1.1 Introduction to Geometric Deep Learning

Traditional deep learning approaches typically rely on something called Euclidean data, which includes text, images, audio, and many other common data types. Data in this format follows the rules of Euclidean geometry, such as the shortest path between 2 points being a straight line. For example, when given data consisting of points with values a and b, the distance between two points can be found using the Pythagorean theorem:

$$d = \sqrt{(a_2 - a_1)^2 + (b_2 - b_1)^2}$$

While this works for many types of data, it cannot be applied to non-Euclidean data. For example, the Pythagorean theorem cannot be used to find the distance between 2 points on a graph. Unlike images or sequences, which follow regular grid-like or linear structures, graphs represent data through nodes and edges, capturing features and relationships in an unordered, interconnected format. To find the distance between two nodes on a graph, you would instead need an algorithm that could find the smallest number of edges to traverse to get from one point to the other. Thus, graphs are considered a type of non-Euclidean data, since the rules of euclidean geometry do not apply to them. Geometric deep learning seeks to find approaches to deep learning that can work on non-Euclidean data. The main focus of this capstone project is a specific type of geometric deep learning: graph learning models.

## 1.2 Graph Machine Learning Tasks

Graph neural networks are designed for a variety of tasks. There are 4 main types of tasks we can solve with graph learning models: Node classification, which is predicting the class of a node, for example if a person in a social network is male or female; link prediction, which is predicting if there is a connection between 2 nodes, such as if two people are friends on Facebook; whole graph learning, which is making some prediction about a whole graph, such as predicting if a molecule is toxic; and community detection, which is clustering nodes into communities.

## 1.3 Limitations of Existing Models

Existing graph learning models have demonstrated significant effectiveness in each of the main task types. However, they have key limitations that affect their performance in specific areas. This paper focuses on limitations common to MPNNs, short for message-passing neural networks, which work by having each node aggregate information from its neighbors. This allows nodes to "pass messages" to each other by reading information from neighboring nodes. In theory, a node can get information from connected nodes any distance away through enough iterations of message passing, though in practice there are limitations that make this difficult.

Oversmoothing is one such limitation. It describes a phenomenon where graph networks perform increasingly worse as more layers are added due to excessive "smoothing". In each layer of a graph neural network, each node's representation will become an aggregate of the representations of its connected nodes. With large numbers of layers, many aggregations are applied before reaching a final output, leading to things being "smoothed" resulting in the loss of more granular information. In many other fields of deep learning performance can be improved by creating deeper networks if there is sufficient data and resources to train them, in graph learning we are limited to only a few layers to avoid oversmoothing.

Oversquashing is another limitation. This phenomenon occurs when important information from distant nodes in a graph is "squashed" as it is passed through the layers of the network. In each layer, nodes aggregate information from their neighbors, as information passes through layers of the network, it gets increasingly compressed due to node embeddings being a fixed size. In a highly connected graph, the number of nodes being aggregated into one node's embedding can increase exponentially, while the size of the node embedding that stores that information is fixed. This results in loss of important details, as the node does not "know" what information will be important for storing later, so it is forced to compress all the information it can into its fixed embedding. This causes MPNNs to have low effectiveness for datasets that require long-range reasoning.

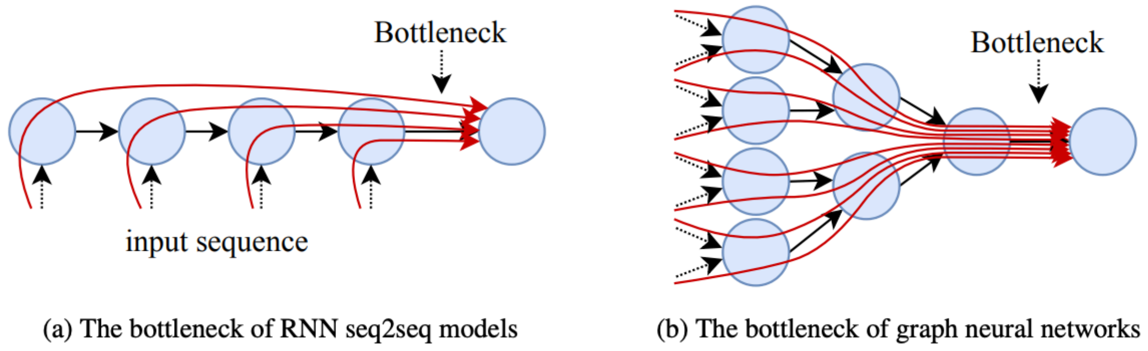
Bottleneck
input sequence
(a) The bottleneck of RNN seq2seq models
Bottleneck
(b) The bottleneck of graph neural networks

Figure 1: Image from "On the bottleneck of graph neural networks and its practical implications" (Alon and Yahav 2021)

Another limitation is in the expressive power of graph learning models. Due to the format in which MPNNs ingest data, they incapable of determining details of the larger structure of the graph, such as its number of connected components. For example, MPNNs would struggle to differentiate between the two graphs below. The model cannot mathematically distinguish when a "loop" has occurred, or how many nodes are in a given connected component.

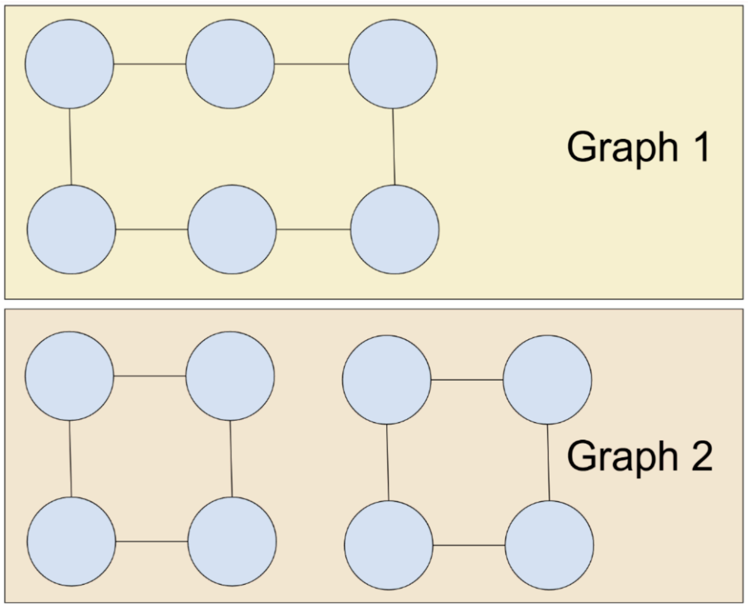

Figure 2: Two example graphs that MPNNs struggle to differentiate.

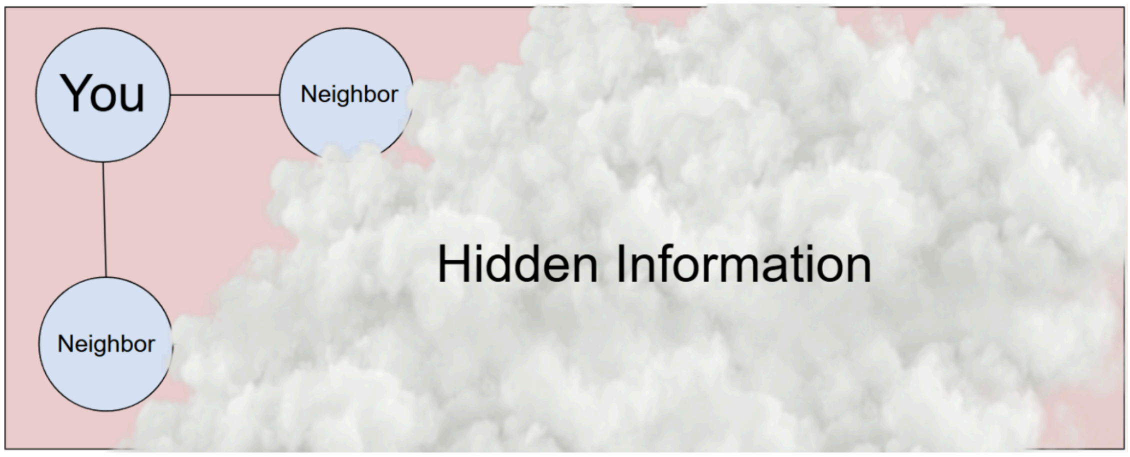

Figure 3: Perspective of the MPNN as each node only has access to its neighbors, which are identical in both Graph 1 and Graph 2.

## 1.4 Related work

Graph Neural Networks(GNNs) and related architectures have seen significant progress this decade. This section highlights key contributions relevant to the concepts and experiments in this paper.

The paper "Semi-Supervised Classification with Graph Convolutional Networks" (Kipf and Welling 2017) introduced the graph convolutional network (GCN). By incorporating graph connectivity and node features, GCNs enable effective semi-supervised learning, demonstrating state-of-the-art performance on several benchmark datasets. This paper established the foundation for subsequent innovations in graph-based learning. It motivates the GCN model using spectral graph convolutions. More details on this in the Methods section.

The paper “How Powerful are Graph Neural Networks?” (Xu et al. 2019)critically examined the expressive power of GNNs, and introduced the Graph Isomorphism Network (GIN). This work demonstrated that GINs can achieve maximal expressiveness among MPNNs, equivalent to the Weisfeiler-Lehman graph isomorphism test.

The paper “Graph Attention Networks” (Veličković et al. 2018) introduced the The Graph Attention Network (GAT), which is an MPNN containing attention mechanisms, enabling the model to adaptively weight each node and edge differently. By increasing the weighting on relevant neighbors the model improves its performance on certain datasets, including Cora.

The paper “A Generalization of Transformer Networks to Graphs” (Dwivedi and Bresson 2021) adapted the original transformer developed for natural language processing to work on arbitrary graphs, creating a model able to better understand long range dependencies in graphs.

The paper “Recipe for a General, Powerful, Scalable Graph Transformer” (Rampášek et al. 2023) introduced a recipe for building a graph transformer that is “general, powerful, and scalable”. The recipe “consists of choosing 3 main ingredients: (i) positional/structural encoding, (ii) local message-passing mechanism, and (iii) global attention mechanism.” More details on this are given in the Methods section.

The paper “Long Range Graph Benchmark” (Dwivedi et al. 2023)introduced 5 graph learning datasets (PascalVOC-SP, COCO-SP, PCQM-Contact, Peptides-func and Peptides-struct) intended to require effective reasoning with long range interactions to perform well on. The paper verified this by showing that transformer models perform significantly better on these datasets than baseline GNNs.

## 1.5 Contribution

In the experiments section of this paper we found evidence for the following major findings:

- GATs are able to reduce, but not fully mitigate, the effects of oversmoothing when increasing the number of layers in a GNN.
- The transformer model graphGPS is able to outperform GCN, GIN, and GAT on a variety of tasks, including one built to require long range reasoning, indicating that graphGPS is able to effectively understand long range connections and use them to make improved predictions.
- GINs are able to perform well on tasks where structural information of the graphs is important, especially in cases where there are no innate node features.
- GATs are able to perform especially well in cases where prominent node features enable them to calculate accurate attention weights.
- GCNs generally perform well on simple tasks but tend to struggle in more complex cases where the GIN and GAT are more performant.

## 2 Methods

### 2.1 Graph Convolutional Networks

One Example of a graph learning model is a GCN, short for graph convolutional network. GCNs often form a basis for more complex models, such as GINs (Graph Isomorphism Networks) and GATs (Graph Attention Networks). GCNs are built to be analogous to traditional convolutional neural networks (CNN for short). Traditional CNNs aggregate information using fixed size kernels with trainable weights. For example, a CNN could compare each pixel in an image to its 8 neighboring pixels using a different weight for the pixel to its right than the pixel to its left. Weighting each of the neighbors differently allows CNN to detect things like vertical vs. horizontal edges. This operation cannot be directly transferred to a graph setting, due to nodes having no canonical order, and variable numbers of neighbors. Because of this, GCNs use graph convolution, which is designed to be permutation invariant and handle an arbitrary number of nodes. A GCN starts by representing each node with an individual node embedding vector. It can be initialized to be any arbitrary vector, common choices include: a detailed set of node features, a single number representing a node's degree, or a constant number for all nodes. Then each layer of the GCN will update the embedding of each node by using some aggregation function on the neighbors of that node. The image below depicts a single node being updated in a GCN with 2 layers.

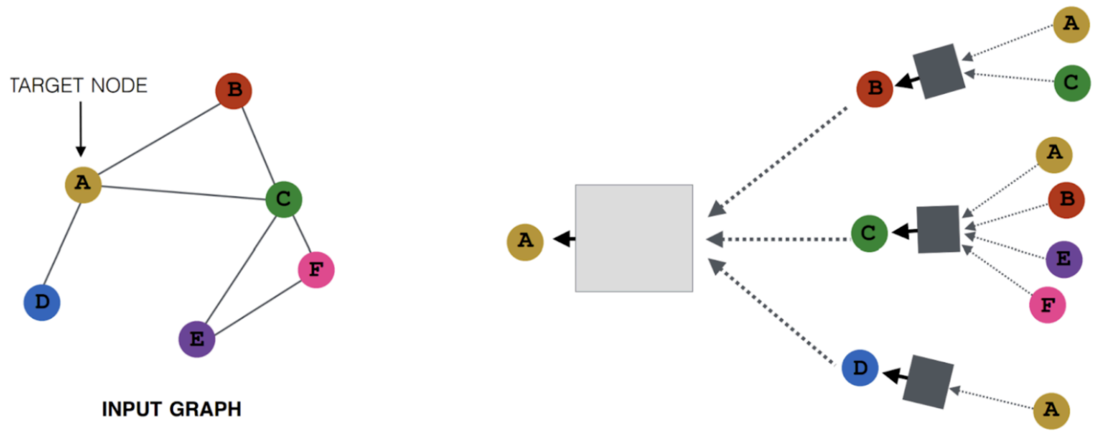

Figure 4: Image from "Representation Learning on Networks" (Hamilton et al. 2018)

In the first layer node B seen in the top right of the image adjusts its node embedding based on the embedding of nodes A and C, likewise all other nodes in the graph adjust their embedding based on their neighbors. Then in the second layer, we see in the middle of the image node A adjusts its embedding based on the current embeddings of its neighbors B, C, and D, which were updated during layer one. Since node A is getting information from node C which had its embedding influenced by nodes E and F, node A is also able to get information from nodes E and F even though it is not directly connected to them. For the aggregation function we can use any function that can take in an arbitrary number of nodes, and is permutation invariant. One popular choice is mean aggregation. Mean aggregation does the following sequence of steps: 1: take the mean of the embeddings of all of a nodes neighbors 2: add this mean to the nodes own embedding, weighted by a learnable weight vector 3: multiply the sum by a learnable weight matrix 4: pass the result through some non-linear activation function such as ReLU. The final value of this becomes the node's new embedding. This is done for every node in the graph for each layer of the network. This process is detailed in this equation, where $W_k$ and $B_k$ are the learnable parameters:

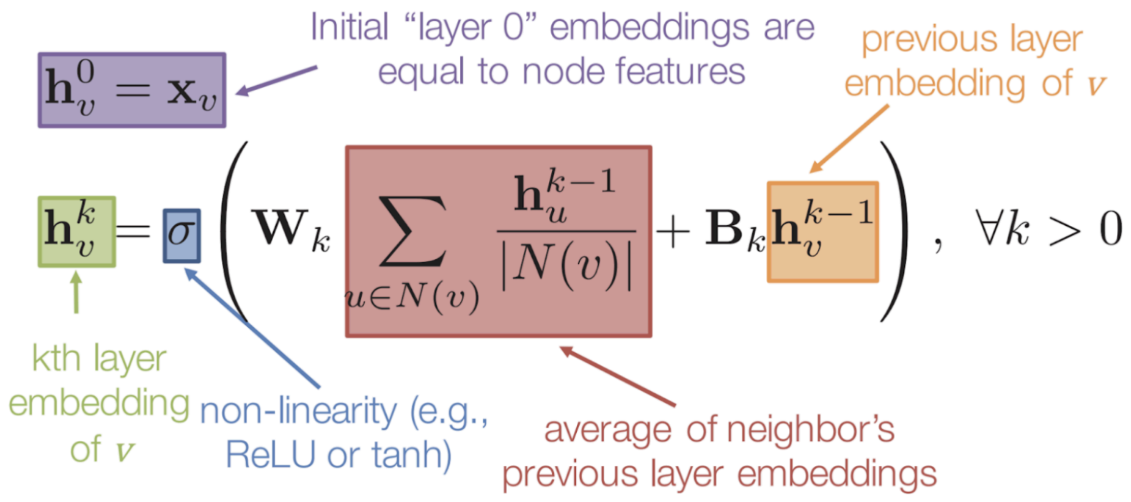
Initial "layer 0" embeddings are equal to node features
$\mathbf{h}_v^0 = \mathbf{x}_v$
previous layer embedding of $v$
$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k > 0$
kth layer embedding of $v$
non-linearity (e.g., ReLU or tanh)
average of neighbor's previous layer embeddings

Figure 5: Image from "Representation Learning on Networks" (Hamilton et al. 2018)

## 2.2 GraphGPS

One example of a graph transformer model is graphGPS (GPS is short for "general, powerful, and scalable"). GraphGPS is a modular framework that uses 3 ingredients to make an effective graph transformer: "(i) positional/structural encoding, (ii) local message-passing mechanism, and (iii) global attention mechanism."

The positional/structural encoding is any method that provides numeric information representing the position or structure of nodes within the graph. Positional encodings represent information about a node's position within the graph; when two nodes are near each other, their positional encodings should closely resemble one another. Structural encodings represent information about a node's structure "when two nodes share similar subgraphs, or when two graphs are similar, their SE should also be close" (Rampášek et al. 2023). These encodings are sub-categorized into three types: local, global, and relative. Local encodings reflect a node's position or structure within a local neighborhood of the graph. Global encodings provide information about a node or graph's overall position and structure. Relative encodings are specific to the relationships between pairs of nodes, capturing pairwise positional or structural differences, such as the distance between two nodes, or if two nodes are in the same cluster.

The local message-passing mechanism can be any message-passing mechanism, such as the GCN, GIN, GAT mentioned in this paper.

The next ingredient is a global attention mechanism. Global attention mechanisms generally work by enabling every element in a system (in this case nodes) to influence every other element's update computation with varying importance. This process relies on learned embeddings for each element $X_n$, as well as three matrices: a query matrix $W_Q$, a key matrix $W_k$, and a value matrix $W_V$. To determine the influence that one element has on another, the mechanism calculates a set of relevance scores that measure the similarity between the query representation of a target element (its corresponding vector in $W_Q$ based on its embedding) and the key representations of all other elements (their vectors in $W_K$). The result is typically divided by the square root of the dimensionality of the key/query vectors to help normalize the output. These relevance scores quantify how strongly each element should contribute to the update of the target element. To normalize the scores relative to each other, a softmax function is applied to all scores. Finally, the weighted contributions are calculated by multiplying these normalized scores with the corresponding value vectors (the vectors in $W_V$) of all elements. The resulting weighted sum provides the updated representation of the target element. Before this process positional encodings are often used to influence a node's embedding, enabling the model to gain information on the relative location of elements when selecting query, key, and value vectors.

Global attention mechanisms can be mathematically expressed as follows:

1. Let $X_n$ denote the embedding of element $n$. Often influenced by some positional encoding.
2. Define the query, key, and value matrices as $W_Q$, $W_K$, and $W_V$, respectively. Compute the query, key, and value representations for an element $n$:

   $$Q_n = W_Q X_n, \quad K_n = W_K X_n, \quad V_n = W_V X_n$$

3. Calculate the relevance scores for element $n$ using a similarity function $f$:

   $$R_{n,m} = f(Q_n, K_m)$$

   where $f(Q_n, K_m)$ is a generic similarity function. $R_{n,m}$ represents the relevance of element $m$ to the element $n$. A common choice is the normalized dot product of the 2 vectors.
4. Obtain the normalized scores using the softmax function:

   $$\alpha_{n,m} = \text{softmax}(R_{n,m})$$

5. Compute the updated representation for element $n$:

   $$U_n = \sum_m \alpha_{n,m} V_m$$

This set of steps forms a baseline for how global attention works and many techniques build further upon or modify it. A variety of global attention mechanisms can be used in the GPS framework, the paper "Recipe for a General, Powerful, Scalable Graph Transformer" (Rampášek et al. 2023) emphasizes the benefits of using a linear global attention mechanism such as Performer (Choromanski et al. 2022) and BigBird (Zaheer et al. 2021) to allow the model to run efficiently on larger graphs. The 3 GPS ingredients (positional/structural encoding, message passing mechanism, and global attention mechanism) are combined simply in its update rule. At each layer, the following steps are applied: step 1, calculate new node and edge features using the message passing mechanism; step 2, calculate new node features using the global attention mechanism; step 3, combine the node features from steps 1 and 2 using an MLP. This is written out mathematically in "Recipe for a General, Powerful, Scalable Graph Transformer" (Rampášek et al. 2023):

$$\begin{aligned} \mathbf{X}^{\ell+1}, \mathbf{E}^{\ell+1} &= \texttt{GPS}^{\ell}\left(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{A}\right) && (1) \\ \text{computed as} \quad \mathbf{X}_M^{\ell+1},\ \mathbf{E}^{\ell+1} &= \texttt{MPNN}_e^{\ell}\left(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{A}\right), && (2) \\ \mathbf{X}_T^{\ell+1} &= \texttt{GlobalAttn}^{\ell}\left(\mathbf{X}^{\ell}\right), && (3) \\ \mathbf{X}^{\ell+1} &= \texttt{MLP}^{\ell}\left(\mathbf{X}_M^{\ell+1} + \mathbf{X}_T^{\ell+1}\right), && (4) \end{aligned}$$

where $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix of a graph with $N$ nodes and $E$ edges; $\mathbf{X}^{\ell} \in \mathbb{R}^{N \times d_\ell}, \mathbf{E}^{\ell} \in \mathbb{R}^{E \times d_\ell}$ are the $d_\ell$-dimensional node and edge features, respectively; $\texttt{MPNN}_e^{\ell}$ and $\texttt{GlobalAttn}^{\ell}$ are instances of an MPNN with edge features and of a global attention mechanism at the $\ell$-th layer with their corresponding learnable parameters, respectively; $\texttt{MLP}^{\ell}$ is a 2-layer MLP block.

Figure 6: Update function (Rampášek et al. 2023)

## 3 Experiments and Results

### 3.1 Setup

For our experiments we use GCN, GIN, and GAT from the pytorch geometric package (Fey and Lenssen 2019), and graphGPS from the graphGPS github repo (Rampášek et al. 2022). In order to compare the strengths and weaknesses of the 4 models in a variety of circumstances, we download 4 datasets. Cora from Planetoid (Yang, Cohen and Salakhutdinov 2016), IMDB and Enzymes from TUDataset (Morris et al. 2020), and Peptide-Struct from Long Range Graph Benchmark (Dwivedi et al. 2023). For each dataset, the data was split into training and testing sets and the models were set up with specific architectures and hyperparameters tailored to performance on the task (e.g., node vs. graph classification). The Adam optimizer was used with a learning rate between 0.001 and 0.01 and varied weight decay to prevent overfitting, trying values from 0 to 5e-4. This combination helped manage model generalization and stability. Each model was trained for up to 300 epochs, with training monitored by accuracy and loss values. All models had their architectures and hyperparameters fine tuned for each task in order to compare each model at its best performance. We will use these models and datasets to setup 3 experiments:

Experiment 1 will be to determine if the attention mechanism in GATs enables them to reduce the effects of oversmoothing on performance. To do this we will compare the performance of GCNs to GATs with 1 to 15 layers on the Cora dataset. If the effect of oversmoothing is diminished in GATs we expect it to see less reduction in performance as the number of layers increases when compared to GCNs.

Experiment 2 will compare the 3 non-transformer models (GCN, GIN, and GAT) on all 4 datasets to gain insight into their performance across a variety of circumstances and establish a set of baselines for experiment 3.

Experiment 3 will run graphGPS on all 4 datasets and compare results to experiment 2. With the goal being to determine if the transformer's ability to effectively use long range connections increases its performance when compared to the non-transformer models. If graphGPS is effectively able to utilize long range connections We expect to see an increase in performance on the Peptide-Struct dataset which is designed to require long range reasoning, as well as potential performance increases on other datasets.

### 3.2 Datasets

In our experiments, we use a diverse set of datasets to present different learning objects and challenges for graph learning. Cora is a citation network dataset where nodes represent academic documents and edges represent citation relationships. The node features are sparse word vectors extracted from document content. This dataset is used for node classification tasks where the goal is to predict the category of each document. IMDB is a binary graph classification dataset, composed of graphs representing movies, with nodes for actors and edges for co-appearances in movies. The classification task is to predict whether the movie is an action or romance movie. This is appropriate for social network relationship analysis. Enzyme is a multi-class graph classification dataset that involves protein structures, where nodes represent atoms and edges representing bonds. The classification task is to assign each graph to one of six protein classes. Lastly, the Peptides-struct dataset is a graph regression dataset focusing on the 3D structural properties of peptides. Nodes in these graphs represent molecular components and edges show their connectivity. In contrast to other classification problems, Peptide-struct calls for graph-level predictions of aggregated 3D features including best-fit plane, sphericity, and inertia. Mean Absolute Error (MAE) is used to quantify performance on Peptides-struct, which is typically selected for molecular property regression datasets. For the other 3 datasets, we use accuracy as the performance metric.

### 3.3 Experiment 1

For this experiment both models had 16 channels in each of the hidden layers. All layers were identical in size except the first and last layers which were adjusted to match the input and output vector sizes. The GAT used 4 attention heads for all layers. Both models were optimized using Adam with a learning rate of 0.01 and a weight decay of 5e-4. All models were trained for 100 epochs. The model GCN used a ReLU nonlinearity after each layer except the final one. The GAT model used an ELU nonlinearity after each layer except the final one. Both models used a log-softmax at the end to determine the final prediction for each node. We plot the results of the experiment in the following graphs: We see that

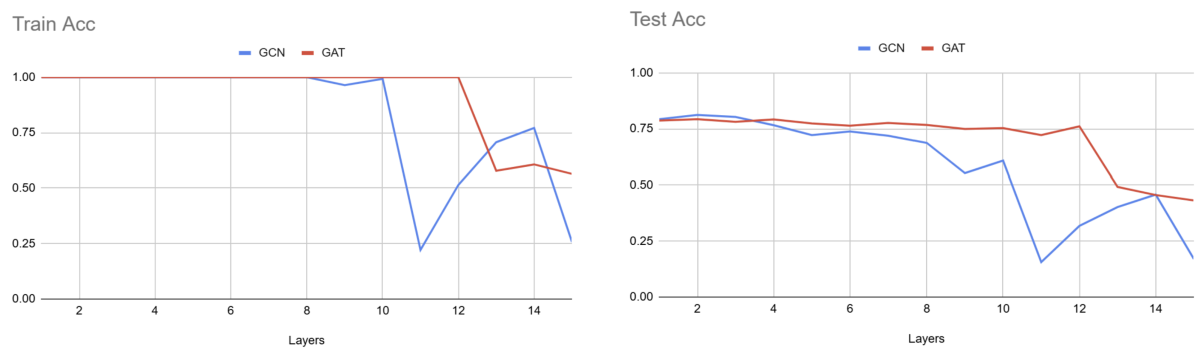

Figure 7: GCN performance vs GAT as number of layers increases.

while GCN slightly outperforms GAT on small numbers of layers, GAT is better able to hold onto its performance as the number of layers increases. We can theorize that the attention mechanism in GATs allows them to focus on more "important" surrounding nodes, reducing the effects of oversmoothing. Both models still see a significant drop in performance as the number of layers increases, indicating that while GAT is able to reduce the effects of oversmoothing it is not able to mitigate them completely. We see that both models are able to maintain high training accuracy longer than they are able to maintain high testing accuracy, we believe the larger model size allows them to better overfit to the training data, counteracting the effect of oversmoothing until it gets too significant.

## 3.4 Experiment 2

For this experiment of comparing GCN, GIN, and GAT across datasets, each model was given an architecture and hyperparameters specific to each dataset. Each model was fine-tuned on each dataset to ensure the models are being compared at their best. The specifications are as follows:

**Cora (Node Classification):**

- **GCN:** 2 GCN layers with 16 hidden channels in each layer, a ReLU between layers 1 and 2, and log softmax applied at the end to turn the results into a normalized distribution across the target classes. Learning rate = 0.01, weigh decay = 5e-4, loss metric = nll_loss, trained for 300 epochs.
- **GIN:** 2 GIN layers with 64 hidden channels in each layer. A ReLu is used between layers 1 and 2 as well as a dropout of 0.5 during training to prevent overfitting to training data. A log softmax is applied after the last layer. Learning rate = 0.01, weigh decay = 5e-4, loss metric = cross_entropy, trained for 300 epochs.
- **GAT:** 4-layer structure with 6 attention heads in layers 1-3 and 1 attention head in layer 4. Each layer has 16 hidden channels for each attention head, each attention head outputs 16 features. A ReLU and dropout of 0.4 is applied after each layer except the last one to introduce nonlinearity and reduce overfitting to training data.

A log softmax is applied after the last layer. Learning rate = 0.01, weight decay = 1e-4, loss metric = cross_entropy, trained for 100 epochs.

**IMDB (Binary Graph Classification):**

- **GCN:** 2 GCN layers with 16 hidden channels in each layer, and 1 fully connected linear layer at the end that maps to the 2 classes. A ReLU is applied between layers 1 and 2 to introduce nonlinearity. A global max pooling is used before the fully connected layer to make a prediction on the full graph. A log softmax is applied after the pooling. Nodes were encoded with their degrees due to the lack of node features. Learning rate = 0.001, weight decay = 5e-4, loss metric = cross_entropy, trained for 100 epochs.
- **GIN:** 2 GIN layers with 16 hidden channels in each layer, and 1 fully connected linear layer at the end that maps to the 2 classes. A ReLU is used between layers 1 and 2 as well as a dropout of 0.5 to introduce a nonlinearity and during training to prevent overfitting to training data. A global mean pooling is used before the fully connected layer to make a prediction on the full graph. A log softmax is applied after the pooling. Nodes were encoded with constant features due to the lack of node features. Learning rate = 0.001, weigh decay = 5e-4, loss metric = cross_entropy, trained for 100 epochs.
- **GAT:** 2 GAT layers with 8 attention heads in both layers. Each layer has 32 hidden channels for each attention head, each attention head outputs 32 features. A ReLU is applied after each layer except the last one to introduce non-linearity. There is 1 fully connected linear layer at the end that maps to the 2 classes. A global mean pooling is used before the fully connected layer to make a prediction on the full graph. A log softmax is applied after the last layer. Nodes were encoded with random features between 0 and 1 due to the lack of node features. Learning rate = 0.001, weight decay = 1e-4, loss metric = cross_entropy, trained for 100 epochs.

**ENZYME (Multi-Class Graph Classification):**

- **GCN:** 2 GCN layers with 16 hidden channels in each layer, and 1 fully connected linear layer at the end that maps to the 6 classes. A ReLU is applied between layers 1 and 2 to introduce nonlinearity. A global max pooling is used before the fully connected layer to max a prediction on the full graph. A log softmax is applied after the pooling. Learning rate = 0.001, weight decay = 5e-4, loss metric = cross_entropy, trained for 100 epochs.
- **GIN:** 2 GIN layers with 16 hidden channels in each layer. A ReLU is used between layers 1 and 2 as well as a dropout of 0.5 to introduce a nonlinearity and during training to prevent overfitting to training data. A global mean pooling is used before the fully connected layer to make a prediction on the full graph. A log softmax is applied after the pooling. Learning rate = 0.001, weigh decay = 5e-4, loss metric = cross_entropy, trained for 100 epochs.
- **GAT:** 2 GAT layers with 8 attention heads in both layers. Each layer has 32 hidden channels for each attention head, each attention head outputs 32 features. A ReLU is applied after each layer except the last one to introduce non-linearity. There is 1 fully connected linear layer at the end that maps to the 2 classes. A global mean

pooling is used before the fully connected layer to make a prediction on the full graph. A log softmax is applied after the last layer. Learning rate = 0.001, weight decay = 5e-4, loss metric = cross_entropy, trained for 100 epochs.

**Peptide-Struct (Graph Regression):**

- **GCN:** 3 GCN layers with 9 hidden channels in each layer, and 1 fully connected linear layer at the end that maps to the 11 numbers being predicted. A ReLU and a dropout of 0.6 is applied after each GCN layer except the last to introduce nonlinearity and reduce overfitting to training data. A global mean pooling is used before the fully connected layer to max a prediction on the full graph. Edges with no encoding are given a constant encoding when found. Learning rate = 0.002, loss metric = mse_loss, trained for 100 epochs.
- **GIN:** 3 GIN layers with 64 hidden channels in the first layer, 128 in the second, and 64 again in the third. 1 fully connected linear layer at the end maps to the 11 numbers being predicted. A ReLU and a dropout of 0.5 is applied after each GIN layer except the last to introduce nonlinearity and reduce overfitting to training data. A global mean pooling is used before the fully connected layer to make a prediction on the full graph. Edges with no encoding are given a constant encoding when found. Learning rate = 0.002, loss metric = mse_loss, trained for 100 epochs.
- **GAT:** 3 GAT layers with 4, 4, and 1 head(s) respectively. The first layer outputs 64 channels per head, the second 128 per head, and the last 64 channels. 1 fully connected linear layer at the end maps to the 11 numbers being predicted. A ReLU and a dropout of 0.5 is applied after each GAT layer except the last to introduce nonlinearity and reduce overfitting to training data. A global mean pooling is used before the fully connected layer to make a prediction on the full graph. Edges with no encoding are given a constant encoding when found. Learning rate = 0.002, loss metric = mse_loss, trained for 100 epochs.

We show the results of running each of these models in the table below:

Table 1: Model performance on different datasets. Training and test accuracies (or mean absolute errors) are reported.

| Dataset | Model | Training Accuracy / Mean Absolute Error | Test Accuracy / Mean Absolute Error |
|---|---|---|---|
| Cora | GCN | 100% | 80.90% |
|  | GIN | 100% | 74.30% |
|  | GAT | 100% | 81.30% |
| IMDB | GCN | 70.75% | 73.00% |
|  | GIN | 67.37% | 77.00% |
|  | GAT | 54.25% | 58.00% |
| ENZYMES | GCN | 39.37% | 31.67% |
|  | GIN | 35.83% | 36.67% |
|  | GAT | 32.50% | 35.83% |
| Peptide-Struct | GCN | 0.4537 (MAE) | 0.4468 (MAE) |
|  | GIN | 0.3990 (MAE) | 0.3949 (MAE) |
|  | GAT | 0.3886 (MAE) | 0.3909 (MAE) |

For the cora dataset we see that of the three models GCN and GAT perform well, and GIN performed more poorly. GAT is able to get a slight edge, likely due to the abundance of node features being useful for calculating accurate attention weights. The poor performance of GIN could be a result of being unable to find the right hyperparameters and architecture for the more complex model, or the Cora dataset could just not require making use of the high graph structure distinguishing capabilities that are GINs main advantage.

For the IMDB dataset GIN performs better than GCN, and GAT performs worse. Performance on a binary graph classification task with no innate node features is highly dependent on a model's ability to distinguish between graph structures. This indicates that GIN is better able to differentiate graph structures compared to GCN and GAT, which follows logically from it having equivalent expressiveness to the Weisfeiler-Lehman graph isomorphism test. The GAT performs particularly weekly compared to the GCN, we believe this could be due to a lack of finding the right hyperparameters or architecture for the more complex model, or it could be that the added complexity of weighting nodes based on attention does not provide any utility in the imdb environment due to the lack of node features.

For the enzymes dataset, we see GIN and GAT perform well, whereas GCN performs more poorly. Like in cora, the abundance of node features are useful for calculating accurate attention weights, enabling GAT to get a high performance. Similar to the imdb dataset, accurately classifying an enzyme is reliant on being able to distinguish its graph structures, which is likely why GIN is able to perform overall the best on this dataset.

For the Peptide-Struct data, we see GAT and GIN perform well, and GCN performs more poorly. GAT has a slight edge over GIN, likely due to the attention mechanisms helping to reduce over squishing for long range connections, as the Peptide-Struct is intended to require long range reasoning. GIN also performs well since this is once again a graph wide task, its high expressive power for graph structures enables it to better understand information about the graph as a whole.

## 3.5 Experiment 3

As mentioned before, our third experiment is running graphGPS on all 4 datasets and comparing results with experiment 2. We want to determine if the transformer's ability to effectively use long range connections increases its performance when compared to the non-transformer models.

**Cora (Node Classification):** The GraphGPS model for the Cora dataset uses 4 layers of GCN + Transformer architecture with 64 hidden units per layer. Each layer applies a ReLU activation function, and a drop out of 0.1 to reduce overfitting. The model uses Laplacian Positional Encoding (LapPE) with 8-dimensional embeddings processed through a 2 layer DeepSet architecture. Node embeddings are generated using a LapPE-based encoder, while edges are unencoded. For optimization, we used AdamW optimizer with a learning rate of 0.0005 and a cosine scheduler. We used cross-entropy loss for the loss function and evaluated the model using accuracy.

**IMDB (Binary Graph Classification):** The GraphGPS model for the IMDB-Binary dataset applies 6 layers of GCN + Transformer architecture with 96 hidden units per layer. Each layer applies a ReLU activation function, and a dropout of 0.1 is used to reduce overfitting.

The model uses Laplacian Positional Encoding (LapPE) with 16-dimensional embeddings and is processed through 2 layer DeepSet architecture. Node embeddings are generated using a linear layer and edges are unencoded. For optimization, we used AdamW optimizer with a learning rate of 0.001, a cosine scheduler, and a weight decay of 0.01. We used cross-entropy loss for the loss function and evaluated using accuracy.

**ENZYME (Multi-Class Graph Classification):** The transformer model for the Enzymes dataset utilizes 6 layers of GCN + Transformer architecture with 128 hidden units per layer. Each layer applies a ReLU activation function, and a dropout of 0.5 is used after each layer to reduce overfitting. The model uses Laplacian Positional Encoding (LapPE) with 16-dimensional embeddings processed through a 2 layer DeepSet architecture. Node embeddings are generated using a linear layer, while edges are unencoded. For optimization, we used AdamW optimizer with a learning rate of 0.001 and a cosine scheduler. We used cross entropy loss for the loss function and evaluated the model using accuracy.

**Peptide-Struct (Graph Regression):** The GraphGPS model for the Peptide-Struct dataset uses a 4-layer CustomGatedGCN + Transformer architecture with 96 hidden units per layer where each layer applies a ReLU activation function, and dropout of 0.0 is utilized to reduce overfitting. The model uses Laplacian Positional Encoding (LapPE) with 16-dimensional embeddings, and used through a 2-layer DeepSet architecture. Node embeddings are made using an Atom+LapPE encoder and the edges are encoded with Bond encoding. For optimization, we used the AdamW optimizer with a learning rate of 0.0003, weight decay of 0.0, and a cosine scheduler with 5 warm-up epochs. The model uses L1 loss for the loss function and is evaluated using Mean Absolute Error (MAE).

Table 2: GraphGPS (Transformer) performance for different datasets.

| Model | Dataset | Training Accuracy / Mean Absolute Error | Test Accuracy / Mean Absolute Error |
|---|---|---|---|
| GraphGPS | Cora | 96.16% | 96.16% |
| GraphGPS | ENZYMES | 43.33% | 36.66% |
| GraphGPS | IMDB | 71.75% | 73.00% |
| GraphGPS | Peptide-Struct | 0.3370 (MAE) | 0.3097 (MAE) |

GraphGPS performed significantly better in Cora, Enzymes, and Peptide-Struct due to their larger and more complex graph structures, richer node and edge features, and the need for capturing global graph relationships. These datasets are able to make use of the global attention used by GraphGPS, which allows the model to outperform simpler models like GIN. In contrast, IMDB's simpler graph structure and lack of node features do not fully leverage the capabilities of GraphGPS. We believe these results, especially those in Peptide-Struct, demonstrate that graphGPS is able to make better use of long range information than GCNs, GINs, and GATs.

## 4 Conclusion

In this study, we examined the performance of GCN, GAT, GIN, and GraphGPS on a diverse range of graph datasets. Our results showcase GraphGPS significantly outperforming the other models in more complex datasets like Cora, Enzymes, and Peptide-Struct, where they have rich node features and crucial global or long range graph relationships. GAT also performed well in these tasks, benefiting from its attention mechanism, while GIN excelled in datasets like IMDB and Enzymes, where graph structure differentiation is key. Overall, our experiments highlight the importance of selecting the right model based on the dataset's structure and features, with GraphGPS offering a more prominent advantage for larger, more intricate graph tasks.

## References

**Alon, Uri, and Eran Yahav.** 2021. "On the Bottleneck of Graph Neural Networks and its Practical Implications." [Link]

**Choromanski, Krzysztof, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller.** 2022. "Rethinking Attention with Performers." [Link]

**Dwivedi, Vijay Prakash, and Xavier Bresson.** 2021. "A Generalization of Transformer Networks to Graphs." [Link]

**Dwivedi, Vijay Prakash, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini.** 2023. "Long Range Graph Benchmark." [Link]

**Fey, Matthias, and Jan Eric Lenssen.** 2019. "Fast Graph Representation Learning with PyTorch Geometric." May. [Link]

**Hamilton, William L., Rex Ying, Jure Leskovec, and Rok Sosic.** 2018. "Representation Learning on Networks." [Link]

**Kipf, Thomas N., and Max Welling.** 2017. "Semi-Supervised Classification with Graph Convolutional Networks." [Link]

**Morris, Christopher, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann.** 2020. "TUDataset: A collection of benchmark datasets for learning with graphs." In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*. [Link]

**Rampášek, Ladislav, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini.** 2022. "Recipe for a General, Powerful, Scalable Graph Transformer." *Advances in Neural Information Processing Systems* 35

**Rampášek, Ladislav, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini.** 2023. "Recipe for a General, Powerful, Scalable Graph Transformer." [Link]

**Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio.** 2018. "Graph Attention Networks." [Link]

**Xu, Keyulu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka.** 2019. "How Powerful are Graph Neural Networks?." [Link]

**Yang, Zhilin, William W. Cohen, and Ruslan Salakhutdinov.** 2016. "Revisiting Semi-Supervised Learning with Graph Embeddings." [Link]

**Zaheer, Manzil, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed.** 2021. "Big Bird: Transformers for Longer Sequences." [Link]