ABSTRACT
Graph Neural Networks (GNNs) have shown success in learning from graph structured data containing node/edge feature information, with application to social networks, recommendation, fraud detection and knowledge graph reasoning. In this regard, various strategies have been proposed in the past to improve the expressiveness of GNNs. For example, one straightforward option is to simply increase the parameter size by either expanding the hidden dimension or increasing the number of GNN layers. However, wider hidden layers can easily lead to overfitting, and incrementally adding more GNN layers can potentially result in over-smoothing. In this paper, we present a model-agnostic methodology, namely Network In Graph Neural Network (NGNN), that allows arbitrary GNN models to increase their model capacity by making the model deeper. However, instead of adding or widening GNN layers, NGNN deepens a GNN model by inserting non-linear feedforward neural network layer(s) within each GNN layer. Although, some works mentioned that adding MLPs within GNN layers could benefit the performance, they did not systematically analyze the reason for the improvement, nor evaluate with numerous GNNs on large-scale graph datasets. In this paper, we demonstrate that NGNN can keep the model stable against either node feature or graph structure perturbations through an analysis of it as applied to a GraphSage base GNN on ogbn-products data. Furthermore, we take a wide-ranging evaluation of NGNN on both node classification and link prediction tasks and show that NGNN works reliably across diverse GNN architectures. For instance, it improves the test accuracy of GraphSage on the ogbn-products by 1.6% and improves the hits@100 score of SEAL on ogbl-ppa by 7.08% and the hits@20 score of GraphSage+Edge-Attr on ogbl-ppi by 6.22%. And at the time of this submission, it achieved two first places on the OGB link prediction leaderboard.

1 INTRODUCTION
Graph Neural Networks (GNNs) capture local graph structure and feature information in a trainable fashion to derive powerful node representations. They have shown promising success on multiple graph-based machine learning tasks [10, 25, 35] and are widely adopted by various web applications including social network [24, 35], recommendation [1, 6, 38], fraud detection [16, 21, 30], etc. Various strategies have been proposed to improve the expressiveness of GNNs [8, 26, 29, 34].

One natural candidate for improving the performance of a GNN is to increase its parameter size by either expanding the hidden dimension or the number of GNN layers. However, this can result in a large computational cost with only a modest performance gain. As a representative example, Figure 1 displays the performance
of GraphSage [8] under different settings on the ogbn-products dataset and the corresponding model parameter sizes. From these results, it can be seen that either increasing the hidden dimension or increasing the number of GNN layers increases the model parameter size exponentially, but brings little performance improvement in terms of test accuracy. For example, in order to improve the accuracy of a 3-layer GraphSage model by 1%, we need to add 2.3× more parameters (by increasing the hidden dimension from 256 to 512). Furthermore, with a larger hidden dimension a model is more likely to overfit the training data. On the other hand, stacking multiple GNN layers may oversmooth the features of nodes [3, 23]. As shown in Figure 1a, GraSeSage reaches its peak performance with only 3 GNN layers and a hidden dimension of 512.

Inspired by the Network-in-Network architecture [20], we present Network-in-Graph Neural-Network (NGNN), a model agnostic methodology that allows arbitrary GNN models to increase their model capacity by making the model deeper. However, instead of adding more GNN layers, NGNN deepens a GNN model by inserting nonlinear feedforward neural network layer(s) within each GNN layer. This leads to a much smaller memory footprint than recent alternative deep GNN architectures [18, 19] and can be applied to all kinds of GNN models with various training methods including full-graph training, neighbor sampling [8], cluster-based sampling [5] and local subgraph sampling [40]. Thus, it can easily scale to large graphs. Moreover, analysis of NGNN in conjunction with GraphSage on perturbed ogbn-products showed that NGNN is a cheap yet effective way to keep the model stable against either node feature or graph structure perturbations.

In this work, we applied NGNN to GCN [14], GraphSage [8], GAT [29] and AGDN [28] and SEAL [40]. We also combine the proposed technique with different mini-batch training methods including neighbor sampling, graph clustering and local subgraph sampling. We conducted comprehensive experiments on several large-scale graph datasets for both node classification and link prediction leading to the following conclusions (which hold as of the time of this submission):

- NGNN improves the performance of GraphSage and GAT and their variants on node classification datasets including ogbn-products, ogbn-archiv, ogbn-proteins and reddit. It improves the test accuracy by 1.6% on the ogbn-products datasets for GraphSage. Furthermore, NGNN with AGDN+BoT+self-KD+C&S [13] achieves the forth place on the ogbn-archiv leaderboard\(^1\) and NGNN with GAT+BoT [32] achieves second place on the ogbn-proteins leaderboard with many fewer model parameters.
- NGNN improves the performance of SEAL, GCN and GraphSage and their variants on link prediction datasets including ogbl-collab, ogbl-ppa and ogbl-ppi. For example, it increases the test hits@100 score by 7.08% on the ogbl-ppa dataset for SEAL, which outperforms all the state-of-the-art approaches on the ogbl-ppa leaderboard\(^2\) by a substantial margin. Furthermore, NGNN achieves an improvement of the test hits@20 score by 6.22% on the ogbl-ppi dataset for GraphSage+EdgeAttr, which also takes the first place on the ogbl-ppi leaderboard.

In summary, we present NGNN, a method that deepens a GNN model without adding extra GNN message-passing layers. We show that NGNN significantly improves the performance of vanilla GNNs on various datasets for both node classification and link prediction. We demonstrate the generality of NGNN by applying them to various GNN architectures.

2 RELATED WORK

Deep models have been widely studied in various domains including computer vision [9, 27], natural language processing [2], and speech recognition [43]. VGG [27] investigates the effect of the convolutional neural network depth on its accuracy in the large-scale image recognition setting. It demonstrates the depth of representations is essential to the model performance. But when the depth grows, the accuracy will not always grow. Resnet [9] eases the difficulties on training the deep model by introducing residual connections between input and output layers. DenseNet [12] takes this idea a step further by adding connections across layers. GPT-3 [2] presents an autoregressive language model with 96 layers that achieves SOTA performance on various NLP tasks. Even so, while deep neural networks have achieved great success in various domains, the use of deep models in graph representation learning is less well-established.

Most recent works [17–19] attempt to train deep GNN models with a large number of parameters and achieved SOTA performance. For example, DeepGCN [18] adapts the concept of residual connections, dense connections, and dilated convolutions [37] to training very deep GCNs. However DeepGCN and its successor DeeperGCN [19] have large memory footprints during model training which can be subject to current hardware limitations. RevGNN [17] explored grouped reversible graph connections to train a deep GNN and has a much smaller memory footprint. However, RevGNN can only work with full-graph training and cluster-based mini-batch training, which makes it difficult to work with other methods designed for large scale graphs such as neighbor sampling [8] and layer-wise sampling [4]. In contrast, NGNN deepens a GNN model by inserting non-linear feedforward layer(s) within each GNN layer. It can be applied to all kinds of GNN models with various training methods including full-graph training, neighbor sampling [8], layer-wise sampling [4] and cluster-based sampling [5].

Xu et al. [34] used Multilayer Perceptrons (MLPs) to learn the injective functions of the Graph Isomorphism Network (GIN) model and showed its effectiveness on graph classification tasks. But they did not show whether adding an MLP within GNN layers works effectively across wide-ranging node classification and link prediction tasks. Additionally, You et al. [36] mentioned that adding

---

\(^1^\)https://ogb.stanford.edu/docs/leader_nodeprop/
\(^2^\)https://ogb.stanford.edu/docs/leader_linkprop/
A graph is composed of nodes and edges \( G = (V, E) \), where \( V = \{v_1, \ldots, v_N\} \) is the set of \( N \) nodes and \( E \subseteq V \times V \) is the set of edges. Furthermore, \( A \in \{0, 1\}^{N \times N} \) denotes the corresponding adjacency matrix of \( G \). Let \( X \in \mathbb{R}^{N \times D} \) be the node feature space such that \( X = (x_1, \ldots, x_N)^T \) where \( x_i \) represents the node feature of \( v_i \). Formally, the \((l + 1)\)-th layer of a GNN is defined as:

\[
h_{\ell+1} = \sigma(f_w(G, h_\ell)), \quad (1)
\]

where the function \( f_w(G, h_\ell) \) is determined by learnable parameters \( w \) and \( \sigma(\cdot) \) is an optional activation function. Additionally, \( h_1 \) represents the embeddings of the nodes in the 1-th layer, and \( h_l = X \) when \( l = 1 \). With an \( L \)-layer GNN, the node embeddings in the last layer \( h_L \) are used by downstream tasks like node classification and link prediction.

### 3.2 Basic NGNN Design

Inspired by the network-in-network architecture [20], we deepen a GNN model by inserting non-linear feedforward neural network layer(s) within each GNN layer. The \((l + 1)\)-th layer in NGNN is thus constructed as:

\[
h_{\ell+1} = \sigma(g_{\text{ngnn}}(f_w(G, h_\ell))). \quad (2)
\]

The calculation of \( g_{\text{ngnn}} \) is defined layer-wise as:

\[
g_{\text{ngnn}}^1 = \sigma(f_w(G, h_1)w^1) \\
\vdots \\
g_{\text{ngnn}}^k = \sigma(g_{\text{ngnn}}^{k-1}w^k)
\]

where \( w_1, \ldots, w_k \) are learnable weight matrices, \( \sigma(\cdot) \) is an activation function, and \( k \) is the number of in-GNN non-linear feedforward neural network layers. The first in-GNN layer takes the output of \( f_w \) as input and performs the non-linear transformation.

### 3.3 Discussion

In this section, we demonstrate that a NGNN architecture can better handle both noisy node features and noisy graph structures relative to its vanilla GNN counterpart.

**Remark 1.** GNNs work well when the input features consist of distinguishable true features and noise. But when the true features are mixed with noise, GNNs can struggle to filter out the noise, especially as the noise level increases.
GNNs follow a neural message passing scheme [7] to aggregate information from neighbors of a target node. In doing so, they can perform noise filtering and learn from the resulting signal when the noise is in some way distinguishable from true features, such as when the latter are mostly low-frequency [22]. However, when the noise level becomes too large and is mixed with true features, it cannot easily be reduced by GNNs [11]. Figure 2 demonstrates this scenario. Here we randomly added Gaussian noise \( \mathcal{N} = N(0, \sigma) \) to node features in ogbn-products data, where \( \sigma \) is the standard deviation ranging from 0.1 to 5.0. We adopt two different methods for adding noise: 1) \( \mathcal{X} = [\mathcal{X} | \mathcal{N}] \) as shown in Figure 2a, where \(|\cdot|\) is a concatenation operation, and 2) \( \mathcal{X} = [\mathcal{X} + \mathcal{N}] \) as shown in Figure 2b. We trained GraphSage models (using the DGL [31] implementation) under five different settings: 1) baseline GraphSage; 2) GraphSage with the hidden dimension increased to 512, denoted as GraphSage-512; 3) 4-layer GraphSage, denoted as GraphSage-4layer; 4) GraphSage with one additional non-linear layer in each GNN layer, denoted as NGNN-GraphSage-1 and 5) GraphSage with two additional non-linear layers in each GNN layer, denoted as NGNN-GraphSage-2. In all cases we used ReLU as the activation function. As shown in Figure 2a, GraphSage performs well when the noise is highly distinguishable from the true features. But the performance starts dropping when the noise is mixed with the true features and decays faster when \( \sigma \) becomes larger than 1.0 as shown in Figure 2b.

The same scenario happens with the gfNN model [22], which is formed by transforming input node features via multiplications of the adjacency matrix followed by an application of a single MLP block. This relatively simple model was shown to be more noise tolerant than GCN and SGC [33]; however, the performance of gfNN turns out to be much lower than the baseline GraphSage model in our experiments, so we do not present results here.

Remark 2. NGNN is a cheap yet effective way to form a GNN architecture that is stable against node feature perturbations.

One potential way to improve the denoising capability of a GNN model is to increase the parameter count via a larger hidden dimension. As shown in Figure 2b, GraphSage-512 does perform better than the baseline GraphSage. But it is also more expensive as its parameter size (675,887) is 3.27× larger than that of baseline GraphSage (206,895). And it is still not as effective as either NGNN model, both of which use considerably fewer parameters (see below) and yet have more stable performance as the noise level increases.

An alternative strategy for increasing the model parameter count is to add more GNN layers. As shown in Figure 2b, adding one more GNN layer, GraphSage-4layer does outperform baseline GraphSage when \( \sigma \) is smaller than 4.0. However, as a deeper GNN potentially aggregates more noisy information from its hop neighbors [39], the performance of GraphSage-4layer drops below baseline GraphSage when \( \sigma \) is 5.0.

In contrast to the above two methods, NGNN-GraphSage achieves much better performance as shown in Figure 2b with fewer parameters (272,687 for NGNN-GraphSage-1 and 338,479 for NGNN-GraphSage-2) than GraphSage-512 and without introducing new GNN layers. It can help maintain model performance when \( \sigma \) is smaller than 1.0 and slow the downward trend when \( \sigma \) is larger than 1.0 compared to the other three counterparts.

Remark 3. NGNN with GNNs can also keep the model stable against graph structure perturbation.

We next provide experimental evidence to show that NGNN works with various GNN architectures for both node classification and link prediction tasks in Sections 4.2 and 4.3. We also show that NGNN works with different training methods in Section 4.4. Finally, we discuss the impact of different NGNN settings in Sections 4.5 and 4.6.

### 4 EXPERIMENTS

We next provide experimental evidence to show that NGNN works with various GNN architectures for both node classification and link prediction tasks in Sections 4.2 and 4.3. We also show that NGNN works with different training methods in Section 4.4. Finally, we discuss the impact of different NGNN settings in Sections 4.5 and 4.6.

#### 4.1 Evaluation Setup

Datasets. We conducted experiments on seven datasets, including ogbn-products, ogbn-arxiv and ogbn-proteins from ogbn [10] and reddit [4] for node classification, and ogbl-collab, ogbl-ppa and ogbl-dblp from ogbl [10] for link prediction. The detailed statistics are summarized in Table 1.

We evaluated the effectiveness of NGNN by applying it to various GNN models including GCN [14], Graphsage [8], Graph Attention Network (GAT) [29], Adaptive Graph Diffusion Networks (AGDN) [28], and SEAL [42] and their variants. Table 2 presents

---

**Table 1: Datasets statistics.**

| Datasets    | # Nodes | # Edges     |
|-------------|---------|-------------|
| Node Classification | | |
| ogbn-products | 2,449,029 | 61,859,140 |
| ogbn-arxiv   | 169,343  | 1,166,243   |
| ogbn-proteins| 132,524  | 39,561,252  |
| reddit       | 232,965  | 114,615,892 |

**Link Prediction**

| Datasets    | # Nodes | # Edges     |
|-------------|---------|-------------|
| ogbl-collab  | 235,868 | 1,285,465   |
| ogbl-ppa     | 576,289 | 30,326,273  |
| ogbl-dblp    | 4,267   | 1,334,889   |

---

The graph of ogbn-product has 61,859,140 edges.

The graph of ogbl-product has 61,859,140 edges.
Table 2: Baseline GNN models used in evaluation.

| GNN Model          | Description                                                                 |
|--------------------|------------------------------------------------------------------------------|
| GraphSage          | Vanilla GraphSage with neighbor sampling.                                    |
| GraphSage-Cluster  | Vanilla GraphSage with cluster based sampling [5].                           |
| GAT-FLAG           | GAT with FLAG [15] enhancement.                                              |
| GAT+BoT            | GAT with bag of tricks [32].                                                 |
| AGDN+BoT           | AGDN with bag of tricks.                                                     |
| AGDN+BoT+self-KD+C&S| AGDN with bag of tricks, knowledge distillation and correct&smooth[13].      |

Table 3: Performance of NGNN on ogbn-products, ogbn-arxiv, ogbn-proteins and reddit.

| Dataset          | Eval Metric | ogbn-products Accuracy(%) | ogbn-arxiv Accuracy(%) | ogbn-proteins Accuracy(%) | reddit Accuracy(%) |
|------------------|-------------|----------------------------|------------------------|---------------------------|--------------------|
| GraphSage        | Vanilla     | 78.27±0.45                 | 71.15±1.66             | 75.67±1.72                | 96.19±0.08         |
|                  | NGNN        | **79.88±1.34**             | **71.77±1.18**         | **76.30±0.96**            | **96.21±0.04**     |
| GraphSage-Cluster| Vanilla     | 78.72±0.63                 | 56.57±1.56             | 67.45±1.21                | 95.27±0.09         |
|                  | NGNN        | 78.91±0.59                 | 56.76±1.08             | **68.12±0.96**            | 95.34±0.09         |
| GAT-NS           | Vanilla     | 79.23±0.16                 | 72.10±1.12             | 81.76±0.17                | 96.12±0.02         |
|                  | NGNN        | 79.67±0.09                 | 71.88±1.10             | 81.91±0.21                | 96.45±0.05         |
| GAT-FLAG         | Vanilla     | 80.75±0.14                 | 71.56±1.11             | 81.81±0.15                | 95.27±0.02         |
|                  | NGNN        | 80.99±0.09                 | 71.74±1.10             | 81.84±0.11                | 95.68±0.03         |

Any score difference between vanilla GNN and NGNN that is greater than 0.5% is highlighted with boldface.

Table 4: Performance (as measured by classification accuracy and ROC-AUC for ogbn-arxiv and ogbn-proteins, respectively) of NGNN combined with bag of tricks on ogbn-arxiv and ogbn-proteins.

| Dataset          | Model                  | Accuracy(%) |
|------------------|------------------------|-------------|
| ogbn-arxiv       | AGDN+BoT               | 74.03±0.15  |
| ogbn-arxiv       | AGDN+BoT+self-KD+C&S   | 74.28±0.13  |
| ogbn-arxiv       | GAT+BoT               | 87.73±0.18  |
| ogbn-proteins    | AGDN+BoT               | 74.25±0.17  |
| ogbn-proteins    | AGDN+BoT+self-KD+C&S   | 74.34±0.14  |
| ogbn-proteins    | GAT+BoT               | 88.09±0.1   |

4.2 Node classification

Firstly, we analyzed how NGNN improves the performance of GNN models on node classification tasks. Table 3 presents the overall results. It can be seen that NGNN-based models outperform their baseline models in most of the cases. Notably, NGNN tends to perform well with GraphSage. It improves the test accuracy of GraphSage on ogbn-products and ogbn-arxiv by 1.61 and 0.62 respectively. It also improves the ROC-AUC score of GraphSage on ogbn-proteins by 0.63. But as the baseline performance of reddit dataset is quite high, not surprisingly, the overall improvement of NGNN is not significant.

We further analysis the performance of NGNN combined with bag of tricks [32] on ogbn-arxiv and ogbn-proteins in Table 4. It can all the baseline models. We directly followed the implementation and configuration of each baseline model from the OGB [10] leaderboard and added non-linear layer(s) into each GNN layer for NGNN.
be seen that NGNN-based models outperform their vanilla counterparts. NGNN with AGDN+BoT+self-KD+C&S even achieves the first place over all the methods with no extension to the input data on the ogbn-arxiv leaderboard as of the time of this submission (The forth place on the entire ogbn-arxiv leaderboard). NGNN with GAT+BoT also achieves the second place on the ogbl-ddi leaderboard with an improvement of hit@20 by 5.47% more, NGNN with GraphSage+EdgeAttr achieves the first place on the ogbl-collab, ogbl-ppa and ogbl-ddi datasets. As shown in the tables, the performance improvement of NGNN over SEAL models is significant. NGNN improves the hit@20, hit@50 and hit@100 of SEAL-DGCNN by 4.72%, 4.67% and 7.08% respectively on ogbl-ppa. NGNN with SEAL-DGCNN achieves the first place over all the methods with no extension to the input data on the ogbn-arxiv leaderboard as of the time of this submission (The forth place on the entire ogbn-arxiv leaderboard). NGNN with GAT+BoT also achieves the second place on the ogbn-proteins leaderboard with 5.83 times fewer parameters compared with the current leading method RevGNN-Wide.6

4.3 Link prediction

Secondly, we analyzed how NGNN improves the performance of GNN models on link prediction tasks. Table 5 presents the results on the ogbl-collab, ogbl-ppa and ogbl-pipi datasets. As shown in the tables, the performance improvement of NGNN over SEAL models is significant. NGNN improves the hit@20, hit@50 and hit@100 of SEAL-DGCNN by 4.72%, 4.67% and 7.08% respectively on ogbl-ppa. NGNN with SEAL-DGCNN achieves the first place on the ogbl-ppa leaderboard with an improvement of hit@100 by 5.82% over the current leading method MLP+CN&RA&A_A. Furthermore, NGNN with GraphSage+EdgeAttr achieves the first place on the ogbl-ddi leaderboard with an improvement of hit@20 by 5.47% over the current leading method vanilla GraphSage+EdgeAttr. As GraphSage+EdgeAttr only provided the performance on ogbl-pipi, we do not compare its performance on other datasets. NGNN also works with GCN and GraphSage on link prediction tasks. As shown in the tables, it improves the performance of GCN and GraphSage in all cases. In particular, it improves the hit@20, hit@50 and hit@100 of GCN by 1.64%, 4.21% and 6.57% respectively on ogbl-ppa.

---

6NGNN -GAT+Bot has 11,740,552 parameters while RevGNN-Wide has 68,471,608 parameters.

7https://github.com/lustoo/OGB_link_prediction

| Table 5: Performance of NGNN on ogbl-collab, ogbl-ppa and ogbl-pipi. We use the hit@20, hit@50 and hit@100 as the evaluation metrics. |
|---------------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| | ogbl-collab | | ogbl-ppa | | ogbl-ddi | |
| Metric (%) | Vanilla GNN | NGNN | Vanilla GNN | NGNN | Vanilla GNN | NGNN |
| SEAL- | hit@20 | 45.76±0.72 | 46.19±0.58 | 16.10±1.85 | 20.82±1.76 | 30.75±2.12 | 31.93±3.00 |
| DGCNN | hit@50 | 54.70±0.49 | 54.82±0.20 | 32.58±1.42 | 37.25±0.98 | 43.99±1.11 | 42.39±3.23 |
| | hit@100 | 60.13±0.32 | 60.70±0.18 | 49.36±1.24 | 56.44±0.99 | 51.25±1.60 | 49.63±3.65 |
| GCN-full | hit@10 | 35.94±1.60 | 36.69±0.82 | 4.00±1.46 | 5.64±0.93 | 47.82±5.90 | 48.22±7.00 |
| | hit@50 | 49.52±0.70 | 51.83±0.50 | 14.23±1.81 | 18.44±1.88 | 79.56±3.83 | 82.56±4.03 |
| | hit@100 | 55.74±0.44 | 57.41±0.22 | 20.21±1.92 | 26.78±0.92 | 87.58±1.33 | 89.48±1.68 |
| GraphSage-full | hit@10 | 32.59±3.56 | 36.83±2.56 | 3.68±1.02 | 3.52±1.24 | 54.27±9.86 | 60.75±4.94 |
| | hit@50 | 51.66±0.35 | 52.62±1.04 | 15.02±1.69 | 15.55±1.92 | 82.18±4.00 | 84.58±1.89 |
| | hit@100 | 56.91±0.72 | 57.96±0.56 | 23.56±1.58 | 24.45±2.34 | 91.94±0.64 | 92.58±0.88 |
| GraphSage+EdgeAttr | hit@20 | - | - | - | - | 87.06±4.81 | 93.28±1.61 |
| | hit@50 | - | - | - | - | 97.98±0.42 | 98.39±0.21 |
| | hit@100 | - | - | - | - | 98.98±0.16 | 99.21±0.08 |

The evaluation metrics used in ogbl leaderboard are hit@50 for ogbl-collab, hit@100 for ogbl-ppa and hit@20 for ogbl-ddi. Any hit score difference between vanilla GNN and NGNN GNN that is greater than 1% is highlighted with boldface.

4.3 Link prediction

Secondly, we analyzed how NGNN improves the performance of GNN models on link prediction tasks. Table 5 presents the results on the ogbl-collab, ogbl-ppa and ogbl-pipi datasets. As shown in the tables, the performance improvement of NGNN over SEAL models is significant. NGNN improves the hit@20, hit@50 and hit@100 of SEAL-DGCNN by 4.72%, 4.67% and 7.08% respectively on ogbl-ppa. NGNN with SEAL-DGCNN achieves the first place on the ogbl-ppa leaderboard with an improvement of hit@100 by 5.82% over the current leading method MLP+CN&RA&A_A. Furthermore, NGNN with GraphSage+EdgeAttr achieves the first place on the ogbl-ddi leaderboard with an improvement of hit@20 by 5.47% over the current leading method vanilla GraphSage+EdgeAttr. As GraphSage+EdgeAttr only provided the performance on ogbl-pipi, we do not compare its performance on other datasets. NGNN also works with GCN and GraphSage on link prediction tasks. As shown in the tables, it improves the performance of GCN and GraphSage in all cases. In particular, it improves the hit@20, hit@50 and hit@100 of GCN by 1.64%, 4.21% and 6.57% respectively on ogbl-ppa.

| Table 6: Test accuracy (%) of GraphSage and GAT with and without NGNN trained with different training methods on ogbn-products. |
|-----------------|-----------------|-----------------|
| Sampling Methods | full-graph | neighbor sampling | cluster-based sampling |
| GraphSage | 78.27 | 78.70 | 78.72 |
| GraphSage-NGNN | 79.88 | 79.11 | 78.91 |
| GAT | 80.75 | 79.23 | 71.41 |
| GAT-NGNN | 80.99 | 79.67 | 76.76 |

| Table 7: Test accuracy (%) of GraphSage and GAT with different number of non-linear layers added into GNN layers on ogbn-products. |
|-----------------|-----------------|
| Model | GraphSage |
| Hidden-size | 128 | 256 | 512 |
| baseline | 77.44 | 78.27 | 79.37 |
| NGNN -1layer | 77.39 | 79.53 | 79.12 |
| NGNN -2layer | 78.79 | 79.88 | 79.94 |
| NGNN -4layer | 78.79 | 79.52 | 79.88 |

| Model | GAT |
| Hidden-size | 64 | 128 | 256 |
| baseline | 68.41 | 79.23 | 75.26 |
| NGNN -1layer | 69.72 | 79.67 | 77.53 |
| NGNN -2layer | 69.86 | 78.26 | 78.76 |
| NGNN -4layer | 69.41 | 78.23 | 78.61 |
We studied the effectiveness of adding multiple non-linear layers. As shown in the table, NGNN-2layer always performed best with all kinds of training methods on ogbn-products. It is worth mentioning that NGNN also works with local subgraph sampling.

Table 7 presents the results. As the table shows, only applying NGNN to the output GNN layer brings little or no benefit. While applying NGNN to hidden and input GNN layers can improve the model performance, especially applying NGNN to hidden layers. It demonstrates that the benefit of NGNN mainly comes from adding additional non-linear layers into the input and hidden GNN layers.

4.4 Effectiveness of Applying NGNN to Different GNN Layers

Finally, we studied the effectiveness of applying NGNN to only the input GNN layer (NGNN-input), only the hidden GNN layers (NGNN-hidden), only the output GNN layer (NGNN-output) and all the GNN layers on the ogbn-products dataset using GraphSage and GAT. The baseline model is a three-layer GNN model. The hidden dimension size is 256 and 128 for GraphSage and GAT respectively. Table 10 presents the results. As the table shows, only applying NGNN to the output GNN layer brings little or no benefit. While applying NGNN to hidden and input GNN layers can improve the model performance, especially applying NGNN to hidden layers. It demonstrates that the benefit of NGNN mainly comes from adding additional non-linear layers into the input and hidden GNN layers.

5 Conclusion and Future Work

We present NGNN, a model agnostic methodology that allows arbitrary GNN models to increase their model capacity by inserting non-linear feedforward neural network layer(s) inside GNN layers. Moreover, unlike existing deep GNN approaches, NGNN does not have large memory overhead and can work with various training methods including neighbor sampling, graph clustering and local subgraph sampling. Empirically, we demonstrate that NGNN can work with various GNN models on both node classification and link prediction tasks and achieve state-of-the-art results. Future work includes evaluating NGNN on more GNN models and investigating whether NGNN can work on broader graph-related prediction tasks. We also plan to explore methodologies to make a single GNN layer deeper in the future.
### Table 11: Model settings of NGNN models. The column of NGNN position presents where we put the non-linear layers. hidden-only means only applying NGNN to the hidden GNN layers, input-only means only applying NGNN to the input layer, all-layer means applying NGNN to all the GNN layers. The column of NGNN setting presents how we organize each NGNN layer. For example, 1-relu+1-sigmoid means NGNN contains one feedforward neural network with ReLU as its activation function followed by another feedforward neural network with Sigmoid as its activation function and 2-relu means NGNN contains two feedforward neural network layers with ReLU as the activation function of each layer.

| Dataset         | Model                | hidden size | layers | aggregation | NGNN position | NGNN setting       |
|-----------------|----------------------|-------------|--------|-------------|----------------|--------------------|
| **Node classification tasks** |                      |             |        |             |                |                    |
| ogbn-product    | GraphSage            | 256         | 3      | mean        | hidden-only    | 1-relu+1-sigmoid   |
| ogbn-product    | GraphSage-cluster    | 256         | 3      | mean        | hidden-only    | 1-relu+1-sigmoid   |
| ogbn-product    | GAT-flag             | 256         | 3      | sum         | hidden-only    | 1-relu+1-sigmoid   |
| ogbn-product    | GAT-ns               | 256         | 3      | sum         | hidden-only    | 1-relu+1-sigmoid   |
| ogbn-axriv      | GraphSage            | 256         | 3      | mean        | hidden-only    | 1-relu+1-sigmoid   |
| ogbn-axriv      | GraphSage-cluster    | 256         | 3      | mean        | hidden-only    | 1-relu+1-sigmoid   |
| ogbn-axriv      | GAT-flag             | 256         | 3      | sum         | hidden-only    | 1-relu+1-sigmoid   |
| ogbn-axriv      | GAT+BoT              | 120         | 6      | sum         | hidden-only    | 2-relu             |
| ogbn-axriv      | AGDN+BoT             | 256         | 3      | GAT-HA      | hidden-only    | 1-relu             |
| ogbn-axriv      | AGDN+BoT+self-KD+C&S  | 256         | 3      | GAT-HA      | hidden-only    | 1-relu             |
| ogbn-protein    | GraphSage            | 256         | 3      | mean        | hidden-only    | 1-relu             |
| ogbn-protein    | GraphSage-cluster    | 256         | 3      | mean        | hidden-only    | 1-relu             |
| ogbn-protein    | GAT-flag             | 256         | 3      | sum         | hidden-only    | 1-relu             |
| ogbn-protein    | GAT-ns               | 256         | 3      | sum         | hidden-only    | 1-relu             |
|reddit           | GraphSage            | 256         | 3      | mean        | hidden-only    | 1-relu+1-sigmoid   |
|reddit           | GraphSage-cluster    | 256         | 3      | mean        | hidden-only    | 1-relu+1-sigmoid   |
|reddit           | GAT-flag             | 256         | 3      | sum         | hidden-only    | 1-relu+1-sigmoid   |
|reddit           | GAT-ns               | 256         | 3      | sum         | hidden-only    | 1-relu+1-sigmoid   |
| **Link prediction tasks** |                  |             |        |             |                |                    |
| ogbl-collab     | Seal-DGCNN           | 256         | 3      | sum         | all-layers     | 1-relu             |
| ogbl-collab     | GCN-full             | 256         | 3      | mean        | hidden-only    | 2-relu             |
| ogbl-collab     | GraphSage-full       | 256         | 3      | mean        | hidden-only    | 2-relu             |
| ogbl-ppa        | Seal-DGCNN           | 32          | 3      | sum         | all-layers     | 1-relu             |
| ogbl-ppa        | GCN-full             | 256         | 3      | mean        | hidden-only    | 2-relu             |
| ogbl-ppa        | GraphSage-full       | 256         | 3      | mean        | hidden-only    | 2-relu             |
| ogbl-ppi        | Seal-DGCNN           | 32          | 3      | sum         | hidden-layers  | 1-relu             |
| ogbl-ppi        | GCN-full             | 256         | 2      | mean        | input-only     | 1-relu             |
| ogbl-ppi        | GraphSage-full       | 256         | 2      | mean        | input-only     | 1-relu             |
| ogbl-ppi        | GraphSage+EdgeAttr   | 512         | 2      | mean        | all-layers     | 2-relu             |

### REFERENCES

[1] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).

[2] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[3] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3438–3445.

[4] Jie Chen, Tengfei Ma, and Caio Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).

[5] Wei-Lin Chiang, Xuaming Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: an efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 257–266.

[6] Wenchu Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The World Wide Web Conference*. 417–426.

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

[8] William L Hamilton, Rex Ying, and Jure Leskovec. 2020. Large graph representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 1025–1035.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[10] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687* (2020).

[11] Binxuan Huang and Kathleen M Carley. 2019. Residual or gate? towards deeper graph neural networks for inductive graph representation learning. *arXiv preprint arXiv:1904.00835* (2019).
This figure "sample-franklin.png" is available in "png" format from:

http://arxiv.org/ps/2111.11638v1