Wide & Deep Learning for Node Classification

Anonymous Authors[1]

Abstract

Wide & Deep, a simple yet effective learning architecture for recommendation systems developed by Google, has had a significant impact in both academia and industry due to its combination of the memorization ability of generalized linear models and the generalization ability of deep models. Graph convolutional networks (GCNs) remain dominant in node classification tasks; however, recent studies have highlighted issues such as *heterophily* and *expressiveness*, which focus on graph structure while seemingly neglecting the potential role of node features. In this paper, we propose a flexible framework GCNIII, which leverages the Wide & Deep architecture and incorporates three techniques: *Intersect memory*, *Initial residual* and *Identity mapping*. We provide comprehensive empirical evidence showing that GCNIII can more effectively balance the trade-off between *over-fitting* and *over-generalization* on various semi- and full- supervised tasks. Additionally, we explore the use of large language models (LLMs) for node feature engineering to enhance the performance of GCNIII in cross-domain node classification tasks. Our implementation is available at `https://anonymous.4open.science/r/GCNIII`.

1. Introduction

Node classification is a machine learning task in graph-structured data analysis (Sen et al., 2008), where the goal is to assign labels to nodes in a graph based on their properties and the relationships between them. While graph convolutional networks (GCNs) (Kipf & Welling, 2017) have achieved great success in node classification due to their strong generalization performance (Xu et al., 2021), some studies have pointed out that message passing neural networks (Gilmer et al., 2017), such as GCNs, have several limitations including *homophily* assumption (Zhu et al., 2020; Luan et al., 2022) and lack of *expressiveness* (Xu et al., 2019). However, recent studies (Ma et al., 2022; Platonov et al., 2023) have found that GCNs can also achieve strong results on heterophilous graphs. Moreover, the latest work (Luo et al., 2024b) indicates that Graph Transformers (GTs) (Ying et al., 2021; Rampášek et al., 2022; Wu et al., 2023; Deng et al., 2024), which are theoretically proven to be more expressive (Zhang et al., 2023a;b), do not outperform GCNs in node classification tasks. In summary, GCNs remain dominant in node classification tasks.

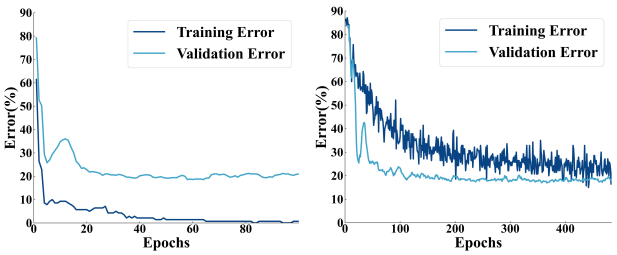

Figure 1. Training error and validation error of the semi-supervised task on Cora with 2-layer vanilla GCN (left) and 64-layer GCNII (right). The training error of deep GCNII is very volatile and much higher than the validation error. We call this phenomenon *over-generalization*.

The classic models, such as GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018) and GraphSAGE (Hamilton et al., 2017), can achieve their best performance with 2-layer shallow models, and stacking more layers will significantly degrade performance. There are at least two possible reasons for this phenomenon. One is *over-smoothing* (Li et al., 2018), in which the embedding vector of the connected nodes becomes indistinguishable after multi-layer graph convolution; the other is that the parameters in the deep graph convolution layers are challenging to optimize (Zhang et al., 2021).

Since shallow GCNs limit their ability to extract information from higher-order neighbors, many studies have explored ways to develop deeper models while relieving the problem of *over-smoothing*. JK-Nets (Xu et al., 2018) use dense skip connections to flexibly leverage different neighborhood ranges. SGC (Wu et al., 2019) removes nonlinearities and collapses weight matrices between consecutive layers by applying the K-th power of the graph convolution matrix

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

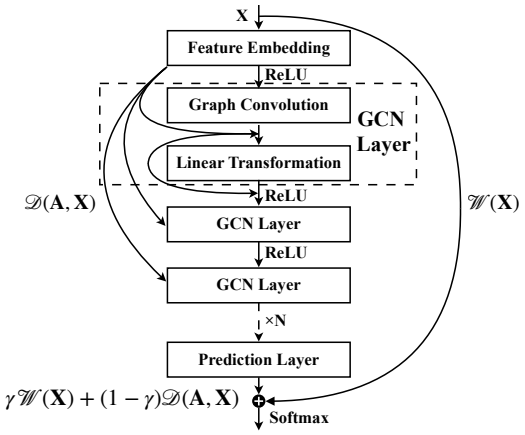

Figure 2. Wide & Deep architecture GCNIII.

in a single layer. DropEdge (Rong et al., 2020) randomly removes a certain number of edges from the input graph at each training epoch, acting like a data augmenter and also a message passing reducer. DAGNN (Liu et al., 2020) decoupling the entanglement of representation transformation and propagation in current graph convolution operations learns graph node representations by adaptively incorporating information from large receptive fields.

When Kipf & Welling (2017) adapt *residual connection* (He et al., 2016) to GCN and PPNP (Gasteiger et al., 2019) uses a variant of personalized PageRank (Page et al., 1999) instead of graph convolution, GCNII (Chen et al., 2020) incorporates ideas from both and continues to achieve state-of-the-art performance to this day. However, when we train 64-layer GCNII on a semi-supervised task, we find that the error on the validation set is much lower than training set as shown in Figure 1, and this is quite different from the performance of 2-layer vanilla GCN which is easy to *over-fitting* on the training set. This phenomenon is often referred to as *under-fitting*, but *under-fitting* models cannot perform well on validation and test sets. This is also not a phenomenon of *over-smoothing*, because *over-smoothing* is global, thus the error of the validation set cannot differ too much from the test set. Therefore, we call this curious phenomenon *over-generalization*, which has not been shown in any other study.

In conclusion, the role and mechanism of deep GCNs are not yet clear. When faced with different graph datas, it remains an open problem what type of GCN, whether shallow or deep, should be used. Unlike many prior works that focused solely on graph structure, we also investigate the role of node features. We propose **G**raph **C**onvolutional **N**etwork with **I**ntersect memory, **I**nitial residual and **I**dentity mapping (GCNIII), a Wide & Deep architecture model as shown in Figure 2 that can more effectively balance the trade-off between *over-fitting* and *over-generalization* and achieves state-of-the-art performance on various semi- and full- supervised tasks.

2. Preliminaries

Node Classification. For node classification tasks, the input data is generally a simple and connected undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n nodes. The information we can use for node classification includes structure information and feature information. Structure information is generally represented by adjacency matrix $\mathbf{A}$ and degree matrix $\mathbf{D}$, and the information of the latter is contained in the former. Feature information is generally represented by node feature matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, which means that each node v is associated with a d-dimensional row vector $\mathbf{x}_v$. The goal is to build model f such that the probability distributions $\mathcal{P}$ of the predicted node classes are as similar as possible to the real labels $\mathcal{C}$:

$$\mathcal{P} = f(\mathbf{A}, \mathbf{X}). \tag{1}$$

However, the two types of information in different datasets are different. Many previous studies, such as *heterophily* and *expressiveness*, focus on structural information, while in-depth studies on feature information are few. For example, the most classic citation network datasets uses sparse features based on a bag-of-words representation of the document. How does the sparsity or denseness of features affect the performance of node classification tasks? Our studies suggest that effectively leveraging both structure and feature information is the key to improving node classification performance.

Wide & Deep. Cheng et al. (2016) suggest that *memorization* and *generalization* are both important for recommender systems. Wide linear models can effectively memorize sparse feature interactions using cross-product feature transformations, while deep neural networks can generalize to previously unseen feature interactions through low-dimensional embeddings. They presented the Wide & Deep learning framework to combine the strengths of both types of model. For a logistic regression problem, the model's prediction is:

$$\Pr(\mathbf{y}=1|\mathbf{x}) = \sigma\left(\mathbf{w}_w^T[\mathbf{x}, \phi(\mathbf{x})] + \mathbf{w}_d^T\mathbf{x}^{(n)} + b\right), \tag{2}$$

where $\mathbf{y}$ is the binary class label, $\sigma(\cdot)$ is the sigmoid function, $\phi(\mathbf{x})$ are the cross product transformations of the original features $\mathbf{x}$, and b is the bias term. $\mathbf{w}_w$ is the vector of all wide model weights, and $\mathbf{w}_d$ are the weights applied on the final embedding $\mathbf{x}^{(n)}$, which is obtained by the iteration $\mathbf{x}^{(l+1)} = ReLU(\mathbf{W}^{(l)}\mathbf{x}^{(l)} + b^{(l)})$ of a feed-forward neural network.

GCN. Kipf & Welling (2017) propose a multi-layer Graph Convolutional Network (GCN) with the following layer-wise propagation rule:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{G}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right). \tag{3}$$

Here, $\tilde{\mathbf{G}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}} = (\mathbf{D}+\mathbf{I}_n)^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I}_n)(\mathbf{D}+\mathbf{I}_n)^{-\frac{1}{2}}$ is the operator corresponding to the *Graph Convolution* in Figure 2, where $\tilde{\mathbf{A}} = \mathbf{A}+\mathbf{I}_n$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections and $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. $\mathbf{W}^{(l)}$ is a layer-specific trainable weight matrix corresponding to the *Linear Transformation* in Figure 2. $\sigma(\cdot)$ denotes the $ReLU(\cdot) = max(0,\cdot)$. $\mathbf{H}^{(l)} \in \mathbb{R}^{n\times d}$ is the matrix of activations in the l-th layer and $\mathbf{H}^{(0)} = \mathbf{X}$ is the node feature matrix.

ResNet. The famous work ResNet from He et al. (2016) solves the difficult problem of training deep neural networks in a very simple way called *residual connection*, which can be formalized as $\mathbf{y} = \mathcal{F}(\mathbf{x}) + \mathbf{x}$. Inspired by He et al. (2016), Kipf & Welling (2017) use *residual connection* between hidden layers to facilitate training of deeper GCN by enabling the model to carry over information from the previous layer's input:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{G}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) + \mathbf{H}^{(l)}. \tag{4}$$

However, simply deepening the network does not bring additional benefits to GCN in node classification tasks.

APPNP. PPNP (Gasteiger et al., 2019) generates predictions for each node based on its own features and then propagates them via the fully personalized PageRank (Page et al., 1999) scheme to generate the final predictions. PPNP's model is defined as $\mathbf{H} = \alpha(\mathbf{I}_n - (1-\alpha)\tilde{\mathbf{G}})^{-1} f_\theta(\mathbf{X})$, where $f_\theta(\cdot)$ denotes a 2-layer MLP. Gasteiger et al. (2019) also proposes a fast approximation variant called APPNP with the following layer-wise propagation rule:

$$\mathbf{H}^{(l+1)} = (1-\alpha)\tilde{\mathbf{G}}\mathbf{H}^{(l)} + \alpha\mathbf{H}^{(0)}, \tag{5}$$

where $\mathbf{H}^{(0)} = f_\theta(\mathbf{X})$. With this propagation rule, we can design very deep models even without using *residual connection* because there are no parameters in the graph convolution layer. This provides a starting point for studying the strong generalization of *Graph Convolution* itself.

GCNII. Chen et al. (2020) propose the GCNII, an extension of the vanilla GCN model with two simple yet effective techniques: *Initial residual* and *Identity mapping*. The idea of *Initial residual* is the same as the propagation rule of APPNP (Gasteiger et al., 2019) and *Identity mapping* is the concept proposed in He et al. (2016), which is a variant of *residual connection*. Unlike Equation (4), the *identity mapping* in GCNII precedes the activation function, which is consistent with the design in ResNet (He et al., 2016). Formally, GCNII's propagation rule is defined as:

$$\mathbf{H}^{(l+1)} = \sigma\left(\left((1-\alpha_l)\tilde{\mathbf{G}}\mathbf{H}^{(l)} + \alpha_l\mathbf{H}^{(0)}\right)\left((1-\beta_l)\mathbf{I}_n + \beta_l\mathbf{W}^{(l)}\right)\right), \tag{6}$$

where $\beta_l = \lambda/l$, α_l and λ are two hyperparameters.

3. GCNIII Model

We propose GCNIII, the first model to extend the Wide & Deep learning to the field of graph-structured data, unifying the effective techniques from previous studies as hyperparameters. We also propose the technical concept of embedding large language models (LLMs) into the framework for upgrading. In all formulas below, $\{\cdot\}$ represents non-essential module that need to be adjusted for different datasets and tasks.

3.1. Wide & Deep Learning

The Wide Component. Generalized linear models with nonlinear feature transformations are widely used for large-scale regression and classification problems with sparse inputs. When we encounter graph-structured data with sparse node features $\mathbf{X} \in \mathbb{R}^{n\times d}$, it is natural to wonder whether linear models can play a role in node classification. We demonstrate the feasibility and validity of the linear models through solid experiments in Appendix C.

Unlike linear regression model in Cheng et al. (2016), the wide component here is a linear classification model of the form:

$$\mathcal{W}(\mathbf{X}) = \{\psi\}(\mathbf{X}\mathbf{W}), \tag{7}$$

where $\mathbf{W} \in \mathbb{R}^{d\times c}$, c is the number of categories of nodes and ψ is the Batch Normalization (Ioffe & Szegedy, 2015). It should be emphasized that the generalization ability of linear models is extremely limited when the amount of data is small. Different from what people are familiar with, although Batch Normalization (Ioffe & Szegedy, 2015) can accelerate convergence and make the model have better classification ability in node classification tasks, it will reduce the generalization performance of the model. We also provide a detailed analysis in Appendix C.

The Deep Component. Compared with the feed-forward neural network in Cheng et al. (2016), *Graph Convolution* can bring more amazing generalization ability improvement (Yang et al., 2023). The deep component is a simple yet flexible GCN model with two core components of *Graph Convolution* and *Linear Transformation*:

$$\mathbf{H}^{(0)} = \{\sigma\}\left(\{\rho\}(\mathbf{X})\mathbf{W_e}\right), \tag{8}$$

$$\mathbf{H}^{(l+1)} = \{\sigma\}\left(\{\tau\}(\tilde{\mathbf{G}})\{\rho\}(\mathbf{H}^{(l)})\{\mathbf{W}^{(l)}\}\right), \tag{9}$$

$$\mathcal{D}(\mathbf{A},\mathbf{X}) = \{\rho\}(\mathbf{H}^{(L)})\mathbf{W_p}. \tag{10}$$

$\mathbf{H}^{(L)}$ is the final layer of propagation. $\mathbf{W_e}$ is the parameter matrix for dimensionality reduction of node features corresponding to *Feature Embedding* in Figure 2 and $\mathbf{W_p}$ is *Prediction Layer*. $\rho(\cdot)$ and $\tau(\cdot)$ are Dropout (Srivastava et al., 2014) and DropEdge (Rong et al., 2020).

Joint Training of Wide & Deep Model. We intend for the two components to be relatively independent, which means that their memorization and generalization abilities are not intertwined, so that we can better understand the sources of model improvement or degradation. Therefore, the output of the model is:

$$\mathcal{P} = Softmax\left(\gamma\mathcal{W}(\mathbf{X}) + (1-\gamma)\mathcal{D}(\mathbf{A},\mathbf{X})\right), \quad (11)$$

where $\mathcal{P}_i$ represents the predicted class distribution for the i-th node. Then we use the cross-entropy loss function and the Adam optimizer (Kingma & Ba, 2015) for joint training.

Although the memorization of the linear models is beneficial for recommendation systems, it requires a large amount of training data as support. When using the wide component for node classification, we should adjust the hyperparameter γ based on the proportion of training set in the datasets and the characteristics of classification tasks, which has strong skills. Moreover, we find that γ cannot be trained as a parameter because the desired generalization ability of the model might not align with the reduction of the loss function value.

3.2. Techniques as Hyperparameters

Intersect memory. We are concerned that when the training data is limited, the poor generalization of the wide component may negatively impact the overall performance of the model. To address this, we propose a technique called *Intersect memory*. The output of the wide component is the distribution of nodes' categories, and we apply a prior attention transformation to this distribution:

$$\mathcal{W}(\mathbf{A},\mathbf{X}) = \mathbf{A}_{\mathbf{IM}}(\{\psi\}(\mathbf{XW})) = \tilde{\mathbf{G}}(\{\psi\}(\mathbf{XW})). \quad (12)$$

The attention matrix between the nodes is the adjacency matrix, allowing this process to be directly performed using *Graph Convolution*. The improved model is still a linear model, but whether to use this technique depends on the datasets.

Initial residual. The prototype of this technique first appeared in Gasteiger et al. (2019), inspired by personalized PageRank (Page et al., 1999), where the authors defined the propagation rule given by Equation (5). From the perspective of *residual connection* (He et al., 2016), Chen et al. (2020) name this technique Initial residual as an improvement to the common residual that carries the information from the previous layer. In this paper, we emphasize that this technique is particularly effective in overcoming *over-smoothing* when designing deep GCNs. We all know *over-smoothing* can degrade model performance, but the difficult-to-train parameters are the root cause of the sudden performance drop as GCNs deepen. Empirical evidence is presented in Appendix D, which supports the viewpoint proposed in Zhang et al. (2021).

Identity mapping. To design the deep GCN model, we need to apply the *Initial residual* technique and remove the parameters $\mathbf{W}^{(l)}$ from Equation (9). However, removing the parameters will inevitably result in the loss of some information. To address this issue, the *Identity mapping* proposed in ResNet (He et al., 2016) can alleviate the challenges of parameter optimization in deep networks. The technique is used in a manner consistent with Equation (6) rather than Equation (4). It is important to note that *Identity mapping* can indeed provide a performance boost in deep GCNs, but the boost is relatively small compared to the large number of parameters added. In GCNIII, *Identity mapping* is an integral component, and *Linear Transformation* is not applied if this technique is not utilized.

3.3. Feature Engineering with LLMs

Although the strong generalization ability of *Graph convolution* is the main reason for GCNs' superior performance in node classification tasks, we emphasize the role of node features in this paper, with detailed experiments presented in Appendix E.

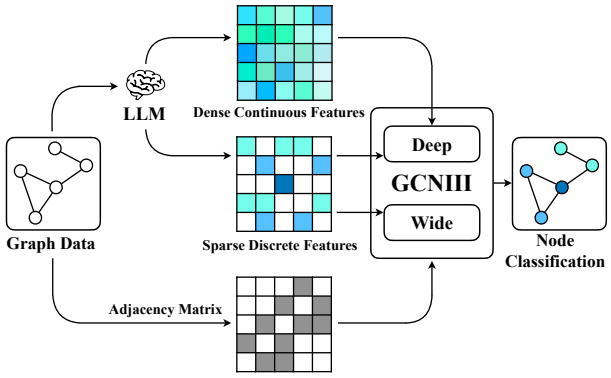

Figure 3. LLM for GCNIII. Sparse discrete features of graph nodes can be constructed using LLM, such as bag-of-words representation of document, which can be used in both the Wide and Deep Components. A unified text-attribute description format can also be used to construct text-attribute graphs (TAGs) as input to the LLM, generating dense continuous features that enhance the learning of the Deep Component.

In the field of node classification tasks, many well-known datasets used in academic research have node characteristics carefully designed by the original authors. When we want to apply the GCNIII model to graph data in other academic or industrial fields, recent research (He et al., 2024) suggests that large language models (LLMs) may be efficient feature encoders, i.e. $\mathbf{X} = \mathbf{LLM}(\mathcal{G})$. Cross-domain graph datas can even be encoded into the same embedding space using unified text-attribute graphs (TAGs) (Liu et al., 2024a), further enhancing the potential of GCNIII as a pre-training foundation model for graphs. The technical concept is illustrated in Figure 3.

4. Over-Generalization

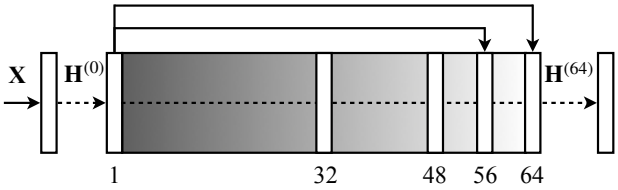

Figure 4. Example network architecture for 64-layer GCNII. The color gradient from black to white represents the weight β_l of the *Linear Transformation* from large to small. *Initial residual* inputs $\mathbf{H}^{(0)}$ directly to each layer, and the network between layers 56 and 64 contains an 8-layer sub-GCNII. This structure is similar to a reversed JKNet.

To this day, GCNII remains one of the most outstanding deep GCN models. Chen et al. (2020) prove that a K-layer GCNII can express a K order polynomial filter $\left(\sum_{\ell=0}^{K}\theta_\ell\tilde{\mathbf{L}}^\ell\right)\mathbf{x}$ with arbitrary coefficients θ, and this is considered a theoretical explanation for the superior performance of GCNII. When training a 64-layer GCNII on the classic citation dataset Cora, we observe an uncommon phenomenon, as shown in Figure 1. We call this phenomenon *over-generalization*, which piques our interest in exploring the cause behind it and revisiting the source of GCNII's SOTA performance.

Dropout is the key. In fact, this can be easily inferred intuitively from Figure 1, because dropout is the only component in the entire end-to-end GCNII model that has a different structure during training and inference. Due to GCNII's complex structure, the authors (Chen et al., 2020) also do not find that dropout has such a significant impact on model performance. Taking the Cora dataset as an example, we find through experimental studies that removing all dropout from GCNII results in a drop in accuracy from over 85% to 82%.. This means that although deep GCNII is theoretically capable of resolving the *over-smoothing* issue, without dropout, its actual performance is no different from a 2-layer GCN. We also find the most critical of all dropout layers is the one before the *Feature Embedding* (shown in Figure 2), and removing the dropout at this position will lead to a noticeable decrease in the model's accuracy. Dropout in Equation (8) is not commonly seen in the design of GCNs, but it is indeed one of the key aspects of the GCNII model. Srivastava et al. (2014) propose dropout as a regularization method by sampling from an exponential number of different "thinned" networks. We argue that the dropout in Equation (8) is more akin to a robust feature selection process, where a subset of features is randomly selected for feature embedding at each epoch. This process enhances the model's ability to efficiently leverage node feature information, thereby improving its generalization performance.

Ultra-deep is not necessary. Simply using dropout is not enough. In the right part of Figure 1, the training error remains much higher than the validation error throughout the training process, while the validation error decreases very quickly. For a 2-layer GCN, no matter how the dropout rate is set, *over-generalization* cannot occur. We propose that a certain model depth is a necessary condition for *over-generalization*, but how deep should GCNII be? We find that an 8-layer GCNII removing *Identity mapping*, which is a variant of APPNP, is sufficient to achieve an average accuracy of 85%. This model has significantly fewer parameters compared to the original 64-layer GCNII, leading to a noticeable improvement in training speed. Our analysis suggests that unlike other deep neural networks, GCNII's power is primarily derived from the layers near the output. As shown in Figure 4, the network from layers 56 to 64 contains the 8-layer GCNII described above, as the β_l of these layers is close to 0. To better understand the effect of model layers, we have the following Theorem.

Theorem 4.1. *Let the K-layer GCNII model be $f_K(\mathbf{A},\mathbf{X})$. $\forall\epsilon>0$, $\exists K_0\in\mathbb{N}^*$ such that when $K>K_0$, we have $\|f_{K+1}(\mathbf{A},\mathbf{X})-f_K(\mathbf{A},\mathbf{X})\|_2<\epsilon$.*

The proof of Theorem 4.1 is in Appendix A. We also find that GCNII cannot contain a linear model, that is, feature information must pass through at least one two-layer MLP with ReLU activation from input to output, which is the motivation for our proposed GCNIII model.

Attention is all your need. We suggests that graph can be viewed as a form of static, discrete self-attention mechanism (Vaswani et al., 2017). The matrix operation form of self-attention is:

$$\text{softmax}\left(\frac{(\mathbf{X}\mathbf{W}^Q)(\mathbf{X}\mathbf{W}^{K^T})}{\sqrt{d_k}}\right)*(\mathbf{X}\mathbf{W}^V). \qquad (13)$$

Graph Convolution $\tilde{\mathbf{G}}$ corresponds to the attention matrix on the left-hand side of Equation (13). Regardless of Identity mapping, *Initial residual* causes the "attention" of GCNII to asymptotically approach $\alpha(\mathbf{I}_n-(1-\alpha)\tilde{\mathbf{G}})^{-1}$ as the number of layers increases indefinitely. Moreover, *Identity mapping* enables the "attention" to fine-tune through data. A conventional attention matrix is typically dense and captures global attention information between elements. We calculate the attention density values for both on Cora with $\alpha=0.1$, i.e., the proportion of non-zero elements in the attention matrix. The former is 0.0018, while the latter is 0.8423, which demonstrates that GCNII's "attention" captures more information, leading to stronger generalization. Through a comparative analysis of misclassified nodes, we also find that 64-layer GCNII has stronger out-of-distribution generalization ability than 2-layer GCN, as detailed in Appendix G.

5. Other Related Work

The research of Graph Neural Networks (GNNs) for node classification is still a hot topic in machine learning. Song et al. (2023) propose ordering message passing into node representations by aligning a central node's rooted-tree hierarchy with its ordered neurons in specific hops. Pei et al. (2024) indicate that the root cause of *over-smoothing* and *over-squashing* is information loss due to heterophily mixing in aggregation. Zheng et al. (2024b) disentangle the graph homophily into label, structural, and feature homophily. Exploration of Graph Transformers in node classification tasks is still ongoing, Wu et al. (2022) propose a Transformer-style model with kernerlized Gumbel-Softmax operator that decreases the complexity to linearity, then Xing et al. (2024) improve Graph Transoformer with collaborative training to prevent the *over-globalizing* problem while keeping the ability to extract valuable information from distant nodes. The impact of the data cannot be ignored, Liu et al. (2024b) study the effect of class imbalance on node classification from a topological paradigm and Luo et al. (2024a) combine GNNs and MLP to efficiently implement Sharpness-Aware Minimization (SAM), enhancing performance and efficiency in Few-Shot Node Classification (FSNC) tasks. In addition to regular offline learning, Zheng et al. (2024a) conduct online evaluation of GNNs to gain insights into their effective generalization capability to real-world unlabeled graphs under test-time distribution shifts. Chen et al. (2024) use LLMs as feature encoders for node classification.

6. Experiments

In this section, we evaluate the performance of GCNIII on a wide variety of open graph datasets. Although LLMs are powerful and have been the focus of recent deep learning research, they are not the focus of this paper. To maintain fairness, we do not use LLMs in the experiments. The hyperparameter details of all models are presented in Appendix B.

6.1. Dataset and Configuration Details.

Dataset. We use all datasets used for evaluating GCNII (Chen et al., 2020) to evaluate GCNIII. For semi-supervised node classification, we utilize three well-known citation network datasets Cora, Citeseer, and Pubmed (Sen et al., 2008), where nodes symbolize documents and edges denote citation relationships. Each node's feature is represented by a bag-of-words representation of the document. For full-supervised node classification, we use web networks Chameleon (Rozemberczki et al., 2021), Cornell, Texas, and Wisconsin (Pei et al., 2020) in addition to the above three datasets, where nodes represent web pages and edges signify hyperlinks connecting them. Similarly, the features of the nodes are derived from the bag-of-words representation of the respective web pages. For inductive learning, we use Protein-Protein Interaction (PPI) networks which contains 24 graphs, where nodes and edges represent proteins and whether there is an interaction between two proteins. Positional gene sets, motif gene sets and immunological signatures are used as features. The node features of these graphs are all sparse and discrete, which are suitable for the wide component of GCNIII. Statistics of the datasets are shown in Table 1.

Table 1. Dataset statistics.

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,732	3,703	6
Pubmed	19,717	44,338	500	3
Chameleon	2,277	36,101	2,325	4
Cornell	183	295	1,703	5
Texas	183	309	1,703	5
Wisconsin	251	499	1,703	5
PPI	56,944	818,716	50	121

Configuration. The experiments are conducted on a Linux server equipped with an Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz, 256GB RAM and 3 NVIDIA A100-SXM4-40GB GPUs. Because of the small size of the datasets, we only use a single GPU to train the models. All models are implemented in PyTorch (Paszke et al., 2019) version 2.2.1, DGL (Wang et al., 2020) version 2.3.0 with CUDA version 12.1 and Python 3.12.7.

6.2. Semi-Supervised Node Classification

Dataset Splitting. In the semi-supervised node classification task, we conduct a conventional fixed split of training/validation/testing (Yang et al., 2016) on the Cora, Citeseer, and Pubmed datasets, with 20 nodes per class for training, 500 nodes for validation and 1,000 nodes for testing. In this experiment, the number of training nodes is small, which can better evaluate the generalization ability of the models.

Classic GNNs are Strong Baselines. Luo et al. (2024b) suggest that the performance of classic GNN models (Kipf & Welling, 2017; Veličković et al., 2018; Hamilton et al., 2017) may be underestimated due to suboptimal hyperparameter configurations; therefore, we use shallow SOTA models GCN (Kipf & Welling, 2017) and GAT (Veličković et al., 2018), as well as deep SOTA models APPNP (Gasteiger et al., 2019) and GCNII (Chen et al., 2020), as baselines. However, we do not directly reuse the metrics reported in Luo et al. (2024b) because we find that Luo et al. (2024b) use some unfair tricks in training. The open-source code of Luo et al. (2024b) shows that test accuracy is calculated

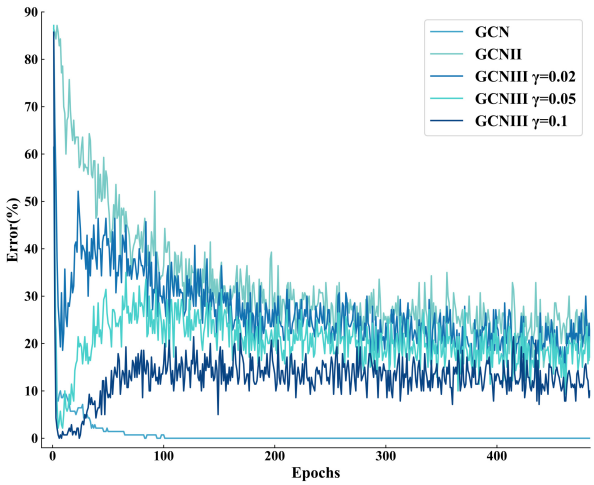

Figure 5. Training error of the semi-supervised task on Cora with GCN, GCNII and GCNIII.

Table 2. Accuracy (%) results on Cora, Citeseer, and Pubmed in the semi-supervised node classification task. The number in parentheses represents the number of layers in the model, and for GCNIII, it indicate the number of layers of the Deep Component model.

Model	Cora	Citeseer	Pubmed
GCN	81.9 ± 0.6 (3)	71.8 ± 0.1 (2)	79.5 ± 0.3 (2)
GAT	80.8 ± 0.6 (3)	69.3 ± 0.8 (3)	78.4 ± 0.9 (2)
APPNP	83.3 ± 0.3 (8)	71.8 ± 0.3 (8)	80.1 ± 0.2 (8)
GCNII	85.2 ± 0.4 (64)	72.8 ± 0.6 (32)	79.8 ± 0.4 (16)
GCNIII	**85.6** ± **0.4** (64)	**73.0** ± **0.5** (16)	**80.4** ± **0.4** (16)

and output for each training epoch, and the model accuracy result is the highest test accuracy of all epochs. Luo et al. (2024b) also randomly repartition the datasets to get "lucky" higher accuracy. Therefore, we re-conduct the experiments in accordance with the optimal model hyperparameters reported in Luo et al. (2024b) under our experimental framework. Details are presented in Appendix F.

Comparison with SOTA. The model implementation in Chen et al. (2020) is based on the PyG library (Fey & Lenssen, 2019). To eliminate the influence of PyG (Fey & Lenssen, 2019) and DGL (Wang et al., 2020) on the model performance, we reproduce the results of APPNP (Gasteiger et al., 2019) and GCNII (Chen et al., 2020) using the hyperparameters in Chen et al. (2020). We train all the models using the same early stopping method in Chen et al. (2020) for fairness. Table 2 reports the mean classification accuracy with the standard deviation of each model after 10 runs. Each run we use a different random seed to ensure that the model is evaluated as fairly as possible. Our experimental results show that GCNIII has improved on the basis of GCNII (Chen et al., 2020), achieving new state-of-the-art the performance on all three datasets.

Table 3. Micro-averaged F1 scores on PPI.

Model	PPI
GraphSAGE (Hamilton et al., 2017)	61.2
GAT (Veličković et al., 2018)	97.3
VR-GCN (Chen et al., 2018)	97.8
Cluster-GCN (Chiang et al., 2019)	99.36
GCNII (Chen et al., 2020)	99.48 ± 0.04
GCNIII	**99.50** ± **0.03**

Over-Generalization of GCNIII Using the Cora dataset as an example, Figure 5 illustrates the training error curves of GCN, GCNII and GCNIII. We believe that the improved training error curve of GCNIII indicates a better balance between the model's fitting ability and generalization, which successfully demonstrates that GCNIII can more effectively balance the trade-off between the *over-fitting* of GCN and the *over-generalization* of GCNII. As γ increases, this balance improves, but it cannot be too large, or it will still lead to *over-fitting*.

6.3. Full-Supervised Node Classification

Following Chen et al. (2020), we evaluate GCNIII in the full-supervised node classification task with 7 datasets: Cora, Citeseer, Pubmed, Chameleon, Cornell, Texas, and Wisconsin. Pei et al. (2020) first randomly split nodes of each class into 60%, 20%, and 20% for training, validation and testing, and measure the performance of all models by the average performance on the test sets over 10 random splits. Chen et al. (2020) follow the criteria for splitting the dataset, so we also adopt the same standard. Besides the previously mentioned models, we also include three variants of Geom-GCN (Pei et al., 2020) as the baseline. We reuse the metrics already reported in Chen et al. (2020) for GCN, GAT, Geom-GCN-I, Geom-GCN-P, Geom-GCN-S and APPNP.

Table 4 reports the mean classification accuracy of each model. We retrain GCNII within our experimental framework using the hyperparameter settings in Chen et al. (2020), however, the results we get on the last four datasets are much lower than those reported in Chen et al. (2020). We observe that GCNIII outperforms GCNII on all 7 datasets, especially the last four *heterophily* datasets, highlighting the superiority of the Wide & Deep GCNIII model. This result suggests that the introduction of the Wide Component linear model enhances GCNIII's predictive power, surpassing the deep-only GCNII model.

6.4. Inductive Learning

Both semi-supervised and full-supervised node classification tasks require that all nodes in the graph are present during training. Hamilton et al. (2017) first propose the

Table 4. Mean classification accuracy of full-supervised node classification.

Model	Cora	Cite.	Pumb.	Cham.	Corn.	Texa.	Wisc.
GCN	85.77	73.68	88.13	28.18	52.70	52.16	45.88
GAT	86.37	74.32	87.62	42.93	54.32	58.38	49.41
Geom-GCN-I	85.19	**77.99**	**90.05**	60.31	56.76	57.58	58.24
Geom-GCN-P	84.93	75.14	88.09	60.90	60.81	67.57	64.12
Geom-GCN-S	85.27	74.71	84.75	59.96	55.68	59.73	56.67
APPNP	87.87	76.53	89.40	54.3	73.51	65.41	69.02
GCNII	88.35 (64)	77.11 (64)	89.58 (64)	54.4 (8)	59.46 (16)	65.68 (32)	65.69 (16)
GCNIII	**88.47** (8)	77.33 (8)	89.88 (32)	**64.69** (2)	**74.59** (2)	**79.73** (2)	**83.33** (3)

Table 5. Ablation study on the Wide Component, where *wide* stands for the linear model in the Wide Component, and the number after "+" indicates the improved accuracy.

Model	Cora	Citeseer	Pubmed
GCN	81.5	71.0	79.2
GCN +*wide*	81.8 +0.3	71.9 +0.9	80.1 +0.9
GAT	83.0	70.4	77.9
GAT +*wide*	83.2 +0.2	70.7 +0.3	78.1 +0.2
APPNP	83.4	71.4	79.9
APPNP +*wide*	83.7 +0.3	71.5 +0.1	80.5 +0.6

inductive learning, which aims to leverage node feature information to efficiently generate node embeddings for previously unseen data. Following Veličković et al. (2018), we use the Protein-Protein Interaction (PPI) dataset for the inductive learning task, with 20 graphs for training, 2 graphs for validation and the rest for testing. We compare GCNIII with the following models: GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018) VR-GCN (Chen et al., 2018), Cluster-GCN (Chiang et al., 2019), and GCNII (Chen et al., 2020). We reuse the metrics reported in Chen et al. (2020), except for GCNII. We re-evaluated GCNII within our experimental framework to better understand the role of the Wide Component in GCNIII. Table 3 indicates that GCNIII outperforms GCNII on PPI, which demonstrates that the memorization ability of the linear model in the Wide Component can also play a role in inductive learning task.

6.5. Ablation Study

Effect of the Wide Component. The three experiments above have confirmed the effectiveness of the Wide Component to some extent, but its initial purpose is to alleviate the *over-generalization* of deep GCNs, so it may not provide benefits to shallow models prone to *over-fitting*. Therefore, we add a basic linear classification model to 2-layer GCN, 2-layer GAT, and 8-layer APPNP as Wide Componet, and set $\gamma = 0.1$. We still perform semi-supervised training on the classical datasets to compare the models. The results in Table 5 show that the Wide Component can still play a role in the shallow models.

Table 6. Ablation study on three techniques, where *memo* stands for *Intersect memory*, *res* stands for *Initial residual*, *map* stands for *Identity mapping* and "-" indicates that the technique is removed.

Model	Cora	Citeseer	Pubmed
GCNIII	85.1	72.8	79.5
GCNIII -*memo*	84.7 -0.4	73.8 +1.0	79.6 +0.1
GCNIII -*res*	63.1 -22.0	29.5 -43.3	51.2 -28.3
GCNIII -*map*	85.7 +0.6	72.7 -0.1	79.4 -0.1

Effect of three techniques. Table 6 presents the results from an ablation study, which assesses the individual contributions of our three techniques: *Intersect memory*, *Initial residual*, and *Identity mapping*. To ensure fairness, we apply the same hyperparameter settings uniformly across the three datasets: $\alpha_l = 0.1$, $\lambda = 0.5$, $\gamma = 0.1$, 64 layers, 64 hidden units, dropout rate of 0.5 and learning rate of 0.01. In this experiment, we fixed the random seed as 42, so the results have a certain randomness. However, we can still conclude that *Initial residual*, as shown in Equation (5), is the most influential factor for the deep GCNs with dropout applied at each layer, while the other two have destabilizing effects and should be applied specifically according to the dataset. Combined with previous experiments, these three techniques are generally beneficial when used properly.

7. Conclusion

In this paper, we find that the training error is much higher than the validation error during the training process when studying the deep GCNII model, and we refer to this phenomenon as *over-generalization*. We conduct an in-depth analysis of this phenomenon and propose GCNIII, the first model to extend the Wide & Deep architecture to graph data. We provide theoretical and empirical evidence that the Wide & Deep GCNIII model more effectively balances the trade-off between over-fitting and over-generalization and achieves state-of-the-art results on various node classification tasks. One meaningful direction for future work is to achieve more efficient node feature representation and graph structure construction by combining LLMs and GCNs models.

Impact Statement

GCNIII is an extension of deep graph convolutional networks and does not infringe upon Google's Wide & Deep model. The application of Large Language Models (LLMs) may raise concerns about transparency and fairness in automated decision-making, potentially exacerbating existing biases. However, we believe that these broader implications align with the ongoing development of AI technologies.

References

Chen, J., Zhu, J., and Song, L. Stochastic training of graph convolutional networks with variance reduction. In *ICML*, 2018.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In *ICML*, 2020.

Chen, Z., Mao, H., Li, H., Jin, W., Wen, H., Wei, X., Wang, S., Yin, D., Fan, W., Liu, H., and Tang, J. Exploring the potential of large language models (llms) in learning on graphs. *arXiv preprint arXiv:2307.03393*, 2024.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., and Shah, H. Wide & deep learning for recommender systems. In *DLRS*, 2016.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*, 2019.

Deng, C., Yue, Z., and Zhang, Z. Polynormer: Polynomial-expressive graph transformer in linear time. In *ICLR*, 2024.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. In *ICLR*, 2019.

Gasteiger, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *ICML*, 2017.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *ICLR*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

He, X., Bresson, X., Laurent, T., Perold, A., LeCun, Y., and Hooi, B. Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning. In *ICLR*, 2024.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

Liu, H., Feng, J., Kong, L., Liang, N., Tao, D., Chen, Y., and Zhang, M. One for all: Towards training one graph model for all classification tasks. In *ICLR*, 2024a.

Liu, M., Gao, H., and Ji, S. Towards deeper graph neural networks. In *KDD*, 2020.

Liu, Z., Qiu, R., Zeng, Z., Yoo, H., Zhou, D., Xu, Z., Zhu, Y., Weldemariam, K., He, J., and Tong, H. Class-imbalanced graph learning without class rebalancing. In *ICML*, 2024b.

Luan, S., Hua, C., Lu, Q., Zhu, J., Zhao, M., Zhang, S., Chang, X.-W., and Precup, D. Revisiting heterophily for graph neural networks. In *ICLR*, 2022.

Luo, Y., Chen, Y., Qiu, S., Wang, Y., Zhang, C., Zhou, Y., Cao, X., and Tang, J. Fast graph sharpness-aware minimization for enhancing and accelerating few-shot node classification. In *NeurIPS*, 2024a.

Luo, Y., Shi, L., and Wu, X.-M. Classic gnns are strong baselines: Reassessing gnns for node classification. In *NeurIPS*, 2024b.

Ma, Y., Liu, X., Shah, N., and Tang, J. Is homophily a necessity for graph neural networks? In *ICLR*, 2022.

Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: Bringing order to the web. In *The Web Conference*, 1999.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. Geom-gcn: Geometric graph convolutional networks. In *ICLR*, 2020.

Pei, H., Li, Y., Deng, H., Hai, J., Wang, P., Ma, J., Tao, J., Xiong, Y., and Guan, X. Multi-track message passing: Tackling oversmoothing and oversquashing in graph learning via preventing heterophily mixing. In *ICML*, 2024.

Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., and Prokhorenkova, L. A critical look at the evaluation of gnns under heterophily: Are we really making progress? In *ICLR*, 2023.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a general, powerful, scalable graph transformer. In *NeurIPS*, 2022.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2020.

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-scale attributed node embedding. *Journal of Complex Networks*, 2021.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, pp. 533–536, 1986.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and EliassiRad, T. Collective classification in network data. *AI magazine*, pp. 29(3), 2008.

Song, Y., Zhou, C., Wang, X., and Lin, Z. Ordered gnn: Ordering message passing to deal with heterophily and over-smoothing. In *ICLR*, 2023.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, pp. 1929–1958, 2014.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NeurIPS*, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *ICLR*, 2018.

Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., and Zhang, Z. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2020.

Wu, F., Zhang, T., de Souza Jr., A. H., Fifty, C., Yu, T., and Weinberger, K. Q. Simplifying graph convolutional networks. In *ICML*, 2019.

Wu, Q., Zhao, W., Li, Z., Wipf, D., and Yan, J. Nodeformer: A scalable graph structure learning transformer for node classification. In *NeurIPS*, 2022.

Wu, Q., Zhao, W., Yang, C., Zhang, H., Nie, F., Jiang, H., Bian, Y., and Yan, J. Sgformer: Simplifying and empowering transformers for large-graph representations. In *NeurIPS*, 2023.

Xing, Y., Wang, X., Li, Y., Huang, H., and Shi, C. Less is more: on the over-globalizing problem in graph transformers. In *ICML*, 2024.

Xu, K., Li, C., Tian, Y., Sonobe, T., ichi Kawarabayashi, K., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *ICLR*, 2019.

Xu, K., Zhang, M., Li, J., Du, S. S., ichi Kawarabayashi, K., and Jegelka, S. How neural networks extrapolate: From feedforward to graph neural networks. In *ICLR*, 2021.

Yang, C., Wu, Q., Wang, J., and Yan, J. Graph neural networks are inherently good generalizers: Insights by bridging gnns and mlps. In *ICLR*, 2023.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform bad for graph representation? In *NeurIPS*, 2021.

Zhang, B., Feng, G., Du, Y., He, D., and Wang, L. A complete expressiveness hierarchy for subgraph gnns via subgraph weisfeiler-lehman tests. In *ICML*, 2023a.

Zhang, B., Luo, S., Wang, L., and He, D. Rethinking the expressive power of gnns via graph biconnectivity. In *ICLR*, 2023b.

Zhang, W., Sheng, Z., Jiang, Y., Xia, Y., Gao, J., Yang, Z., and Cui, B. Evaluating deep graph neural networks. *arXiv preprint arXiv:2108.00955*, 2021.

Zheng, X., Song, D., Wen, Q., Du, B., and Pan, S. Online gnn evaluation under test-time graph distribution shifts. In *ICLR*, 2024a.

Zheng, Y., Luan, S., and Chen, L. What is missing in homophily? disentangling graph homophily for graph neural networks. In *NeurIPS*, 2024b.

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. In *NeurIPS*, 2020.

A. Proof of Theorem 4.1

As we explained in Equation (6), GCNII's core propagation rule is defined as:

$$\mathbf{H}^{(l+1)}=\sigma\left(\left((1-\alpha_l)\tilde{\mathbf{G}}\mathbf{H}^{(l)}+\alpha_l\mathbf{H}^{(0)}\right)\left((1-\beta_l)\mathbf{I}_n+\beta_l\mathbf{W}^{(l)}\right)\right). \tag{14}$$

In the actual implementation of GCNII, $\alpha_l=\alpha\in(0,1)$ is a constant. To reduce the complexity of the notation, we define $\mathbf{W}_{\mathbf{I}}^{(l)}=(1-\beta_l)\mathbf{I}_n+\beta_l\mathbf{W}^{(l)}$, *Feature Embedding* parameter as $\mathbf{W_e}$ and *Prediction Layer* parameter as $\mathbf{W_p}$. As $l\to\infty$, $\mathbf{W}_{\mathbf{I}}^{(l)}\to\mathbf{I}_n$, since $\beta_l=\frac{\lambda}{l}\to 0$. ReLU operation σ in Equation (14) is difficult to handle in analysis. We adopt the same assumption as in Chen et al. (2020), that is, the input node feature vectors are all non-negative. Furthermore, we may assume that the parameters of each layer can map non-negative inputs to non-negative outputs. Therefore, we remove ReLU operation in the subsequent analysis, and the simplified propagation rule is:

$$\mathbf{H}^{(l+1)}=\left((1-\alpha)\tilde{\mathbf{G}}\mathbf{H}^{(l)}+\alpha\mathbf{H}^{(0)}\right)\mathbf{W}_{\mathbf{I}}^{(0)}.$$

Initial $\mathbf{H}^{(0)}=\mathbf{X}\mathbf{W}_e$ is propagated layer by layer

$$\mathbf{H}^{(1)}=(1-\alpha)\tilde{\mathbf{G}}\mathbf{H}^{(0)}\mathbf{W}_{\mathbf{I}}^{(0)}+\alpha\mathbf{H}^{(0)}\mathbf{W}_{\mathbf{I}}^{(1)},$$

$$\mathbf{H}^{(2)}=(1-\alpha)^2\tilde{\mathbf{G}}^2\mathbf{H}^{(0)}\mathbf{W}_{\mathbf{I}}^{(0)}\mathbf{W}_{\mathbf{I}}^{(1)}+(1-\alpha)\alpha\tilde{\mathbf{G}}\mathbf{H}^{(0)}\mathbf{W}_{\mathbf{I}}^{(0)}\mathbf{W}_{\mathbf{I}}^{(1)}+\alpha\mathbf{H}^{(0)}\mathbf{W}_{\mathbf{I}}^{(1)},$$

$$\cdots\cdots$$

Assuming the model has K layers, we can express the final representation as:

$$\mathbf{H}^{(K)}=(1-\alpha)^K\tilde{\mathbf{G}}^K\mathbf{H}^{(0)}\prod_{l=0}^{K-1}\mathbf{W}_{\mathbf{I}}^{(l)}+\alpha\sum_{i=0}^{K-1}\left((1-\alpha)^i\tilde{\mathbf{G}}^i\mathbf{H}^{(0)}\prod_{k=K-i-1}^{K-1}\mathbf{W}_{\mathbf{I}}^{(k)}\right). \tag{15}$$

In Section 4, we analysis 64-layer GCNII with $\alpha=0.1$, then $(1-\alpha)^{64}\approx 0.001$. This means that the 64-layer model represented by the first part on the right of Equation (15) hardly works, and parameters with so many layers are difficult to optimize using the back-propagation algorithm (Rumelhart et al., 1986). From the second part on the right of Equation (15), we can find that the smaller i is, the larger $(1-\alpha)^i$ becomes, the deeper parameter layer is, and the closer these parameters are to $\mathbf{I}_n$.

Surprisingly, the GCNII model explicitly combines all k-layer GCNs ($k=1,2,...,64$), but it is the shallow models that first come into play, and these models are actually located in the deeper layers, closer to the output, rather than near the input. The above analysis focuses on the inference phase of the model; however, during the training phase of the model, the impact of dropout cannot be ignored, and we consider it a direction for future research.

We assume that $f_K(\mathbf{A},\mathbf{X})$ and $f_{K+1}(\mathbf{A},\mathbf{X})$ share the same parameter $\mathbf{W_e}$ and $\mathbf{W_p}$; otherwise, the randomness and complexity of the parameters would inevitably introduce errors. In fact, the focus of our analysis here s the depth of the model, so this assumption is relatively reasonable. $\|\mathbf{A}\|_2=(\lambda_{\mathbf{A}^T\mathbf{A}})^{\frac{1}{2}}$ is the l_2-induced norm or spectral norm, where $\lambda_{\mathbf{A}^T\mathbf{A}}$ denotes the largest eigenvalue of $\mathbf{A}^T\mathbf{A}$. It's very easy to prove that $\|\cdot\|_2$ is a consistent matrix norm, i.e., $\|\mathbf{AB}\|_2\leqslant\|\mathbf{A}\|_2\|\mathbf{B}\|_2$. We also assume that the width of the model is limited. To be more precise, $\mathbf{W}_{\mathbf{I}}^{(l)}\in\mathbb{R}^{n\times n}$ and n is typically in the range of tens or hundreds in practical implementations. In the following, we use $\mathbf{I}$ instead of $\mathbf{I}_n$.

The spectral norm is the maximum singular value of a matrix, and for a symmetric matrix, the spectral norm is equal to the absolute value of its largest eigenvalue. $\tilde{\mathbf{G}}=\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}=(\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})(\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}$ is a symmetric positive semidefinite matrix. The maximum eigenvalue of $\mathbf{A}+\mathbf{I}$ does not exceed $\max(\mathbf{D}_{ii}+1)$, while $(\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}$ normalizes the eigenvalue to the range $[0,1]$, then it is easy to deduce that $\|\tilde{\mathbf{G}}\|_2\leqslant 1$.

Theorem A.1. *(Wu et al., 2019) Let $\mathbf{A}$ be the adjacency matrix of an undirected, weighted, simple graph $\mathcal{G}$ without isolated nodes and with corresponding degree matrix $\mathbf{D}$. Let $\tilde{\mathbf{A}}=\mathbf{A}+\gamma\mathbf{I}$, such that $\gamma>0$, be the augmented adjacency matrix with corresponding degree matrix $\tilde{\mathbf{D}}$. Also, let λ_1 and λ_n denote the smallest and largest eigenvalues of $\mathbf{\Delta}_{sym}=\mathbf{I}-\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$; similarly, let $\tilde{\lambda}_1$ and $\tilde{\lambda}_n$ be the smallest and largest eigenvalues of $\tilde{\mathbf{\Delta}}_{sym}=\mathbf{I}-\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$. We have that*

$$0=\lambda_1=\tilde{\lambda}_1<\tilde{\lambda}_n<\lambda_n. \tag{16}$$

Using the properties of the *Rayleigh quotient*, we can easily prove that the range of eigenvalues of $\tilde{\mathbf{L}} = \mathbf{I} - \tilde{\mathbf{G}}$ is $[0, 2]$, and combining with the Theorem A.1 proposed by Wu et al. (2019), we can further get $\|\mathbf{I} - \tilde{\mathbf{G}}\|_2 < 2$.

In the Adam optimizer (Kingma & Ba, 2015), the regularization term $\lambda_w \|\Theta\|^2$ for weight decay is added to the loss function $\mathcal{L}$ to encourage the model to use smaller weights, thereby reducing overfitting. Therefore, we assume that all parameters have a upper bound, i.e., $\|\mathbf{W}\|_2 < c_1$. Due to the existence of weight decay, the model parameters cannot grow indefinitely, and sparsity characteristics will emerge. Since $\lim_{k\to\infty} \mathbf{W}_{\mathbf{I}}^{(k)} = \mathbf{I}$, we further impose a stronger assumption that the product of any number of *Identity mapping* parameters is bounded above, i.e., $\prod_{i\leqslant k\leqslant j} \mathbf{W}_{\mathbf{I}}^{(k)} < C$. We use $\mathbf{W}_{\mathbf{I}}^{(l)}$ and $\tilde{\mathbf{W}}_{\mathbf{I}}^{(l)}$ to represent the *Identity mapping* parameters of the layers in $f_K(\mathbf{A}, \mathbf{X})$ and $f_{K+1}(\mathbf{A}, \mathbf{X})$, respectively. It is important to emphasize that these parameters are misaligned equality, i.e., $\mathbf{W}_{\mathbf{I}}^{(l)} = \tilde{\mathbf{W}}_{\mathbf{I}}^{(l+1)}$, as our analysis above shows that GCNII is primarily influenced by the layers closer to the output. The input node feature matrix is usually very sparse, and even for dense matrix, each row is normalized so we assume that X also has a small upper bound, i.e., $\|\mathbf{X}\|_2 < c_2$. Furthermore, we have $\|\mathbf{H}^{(0)}\|_2 \leqslant \|\mathbf{X}\|_2 \cdot \|\mathbf{W_e}\|_2 < c_1 c_2$.

Proof. By Equation (15), we obtain:

$$\begin{aligned}
\mathbf{H}^{(K+1)} &= (1-\alpha)^{K+1}\tilde{\mathbf{G}}^{K+1}\mathbf{H}^{(0)}\prod_{l=0}^{K}\tilde{\mathbf{W}}_{\mathbf{I}}^{(l)} + \alpha\sum_{i=0}^{K}\left((1-\alpha)^i\tilde{\mathbf{G}}^i\mathbf{H}^{(0)}\prod_{k=K-i}^{K}\tilde{\mathbf{W}}_{\mathbf{I}}^{(k)}\right) \\
&= (1-\alpha)^{K+1}\tilde{\mathbf{G}}^{K+1}\mathbf{H}^{(0)}\prod_{l=0}^{K}\tilde{\mathbf{W}}_{\mathbf{I}}^{(l)} + \alpha(1-\alpha)^K\tilde{\mathbf{G}}^K\mathbf{H}^{(0)}\prod_{l=0}^{K}\tilde{\mathbf{W}}_{\mathbf{I}}^{(l)} \\
&+ \alpha\sum_{i=0}^{K-1}\left((1-\alpha)^i\tilde{\mathbf{G}}^i\mathbf{H}^{(0)}\prod_{k=K-i}^{K}\tilde{\mathbf{W}}_{\mathbf{I}}^{(k)}\right)
\end{aligned}$$

We first consider:

$$\begin{aligned}
&\|\alpha\sum_{i=0}^{K-1}\left((1-\alpha)^i\tilde{\mathbf{G}}^i\mathbf{H}^{(0)}\prod_{k=K-i}^{K}\tilde{\mathbf{W}}_{\mathbf{I}}^{(k)}\right) - \alpha\sum_{i=0}^{K-1}\left((1-\alpha)^i\tilde{\mathbf{G}}^i\mathbf{H}^{(0)}\prod_{k=K-i-1}^{K-1}\mathbf{W}_{\mathbf{I}}^{(k)}\right)\|_2 \\
&= \alpha\|\sum_{i=0}^{K-1}\left((1-\alpha)^i\tilde{\mathbf{G}}^i\mathbf{H}^{(0)}(\prod_{k=K-i}^{K}\tilde{\mathbf{W}}_{\mathbf{I}}^{(k)} - \prod_{k=K-i-1}^{K-1}\mathbf{W}_{\mathbf{I}}^{(k)})\right)\|_2 \\
&\leqslant \alpha\sum_{i=0}^{K-1}(1-\alpha)^i\|\tilde{\mathbf{G}}\|_2^{\,i}\cdot\|\mathbf{H}^{(0)}\|_2\cdot\|\prod_{k=K-i}^{K}\tilde{\mathbf{W}}_{\mathbf{I}}^{(k)} - \prod_{k=K-i-1}^{K-1}\mathbf{W}_{\mathbf{I}}^{(k)}\|_2 = 0
\end{aligned}$$

Then we consider:

$$\begin{aligned}
&\|(1-\alpha)^{K+1}\tilde{\mathbf{G}}^{K+1}\mathbf{H}^{(0)}\prod_{l=0}^{K}\tilde{\mathbf{W}}_{\mathbf{I}}^{(l)} + \alpha(1-\alpha)^K\tilde{\mathbf{G}}^K\mathbf{H}^{(0)}\prod_{l=0}^{K}\tilde{\mathbf{W}}_{\mathbf{I}}^{(l)} - (1-\alpha)^K\tilde{\mathbf{G}}^K\mathbf{H}^{(0)}\prod_{l=0}^{K-1}\mathbf{W}_{\mathbf{I}}^{(l)}\|_2 \\
&= \|(1-\alpha)^{K+1}\tilde{\mathbf{G}}^{K+1}\mathbf{H}^{(0)}\prod_{l=1}^{K}\tilde{\mathbf{W}}_{\mathbf{I}}^{(l)}(\tilde{\mathbf{W}}_{\mathbf{I}}^{(0)} - \mathbf{I} + \mathbf{I}) + \alpha(1-\alpha)^K\tilde{\mathbf{G}}^K\mathbf{H}^{(0)}\prod_{l=1}^{K}\tilde{\mathbf{W}}_{\mathbf{I}}^{(l)}(\tilde{\mathbf{W}}_{\mathbf{I}}^{(0)} - \mathbf{I} + \mathbf{I}) \\
&- (1-\alpha)^K\tilde{\mathbf{G}}^K\mathbf{H}^{(0)}\prod_{l=0}^{K-1}\mathbf{W}_{\mathbf{I}}^{(l)}\|_2 < (1-\alpha)^{K+1}c_1c_2C\|\tilde{\mathbf{W}}_{\mathbf{I}}^{(0)} - \mathbf{I}\| + \alpha(1-\alpha)^K c_1c_2C\|\tilde{\mathbf{W}}_{\mathbf{I}}^{(0)} - \mathbf{I}\| \\
&+ (1-\alpha)^{K+1}c_1c_2C\|\tilde{\mathbf{G}} - I\|_2 < (1-\alpha)^K c_1c_2C(c_1 + 3 - 2\alpha)
\end{aligned}$$

$\forall\epsilon$, let $(1-\alpha)^K c_1c_2C(c_1+3-2\alpha) = \epsilon/c_1$, we obtain $K = \frac{\log(\epsilon/c_1^2c_2C(c_1+3-2\alpha))}{\log(1-\alpha)}$.

Set $K_0 = \left\lfloor \frac{\log(\epsilon/c_1c_2C(c_1+3-2\alpha))}{\log(1-\alpha)} \right\rfloor$, then when $K > K_0$, we have:

$$\|f_{K+1}(\mathbf{A}, \mathbf{X}) - f_K(\mathbf{A}, \mathbf{X})\|_2 \leqslant \|\mathbf{H}^{(K+1)} - \mathbf{H}^{(K)}\|_2 \cdot \|\mathbf{W_p}\|_2 < \epsilon$$

□

B. Hyperparameters Details

Table 7 summarizes the training configuration of all model for semi-supervised. L_{2_a} denotes the weight decay for *Feature Embedding* and *Prediction Layer*. L_{2_b} denotes the weight decay for *Linear Transformation*. As defined in Chen et al. (2020),the relation between β_l in Equation (6) and λ is $\beta_l = \lambda/l$. The 0/1 in the list of techniques indicates whether *Intersect memory*, *Initial residual* and *Identity mapping* are used.

Table 7. The hyperparameters for Table 2.

Dataset	Model	Hyperparameters
Cora	GCN	layers: 3, lr: 0.001, hidden: 512, dropout: 0.7, L_2: 0.0005
	GAT	layers: 3, lr: 0.001, hidden: 512, dropout: 0.2, L_2: 0.0005
	APPNP	layers:8, lr: 0.01, hidden: 64, α: 0.1, dropout: 0.5, L_2: 0.0005
	GCNII	layers: 64, lr: 0.01, hidden: 64, α_ℓ: 0.1, λ: 0.5, dropout: 0.6, L_{2_a}: 0.01, L_{2_b}: 0.0005
	GCNIII	layers: 64, lr: 0.01, hidden: 64, α_ℓ: 0.1, λ: 0.5, γ: 0.02, dropout: 0.6, L_{2_a}: 0.01, L_{2_b}: 0.0005, techniques: [1, 1, 1]
Citeseer	GCN	layers: 2, lr: 0.001, hidden: 512, dropout: 0.5, L_2: 0.0005
	GAT	layers: 3, lr: 0.001, hidden: 256, dropout: 0.5, L_2: 0.0005
	APPNP	layers: 8, lr: 0.01, hidden: 64, α: 0.1, dropout: 0.5, L_2: 0.0005
	GCNII	layers: 32, lr: 0.01, hidden: 256, α_ℓ: 0.2, λ: 0.6, dropout: 0.7, L_{2_a}: 0.01, L_{2_b}: 0.0005
	GCNIII	layers: 16, lr: 0.01, hidden: 256, α_ℓ: 0.1, λ: 0.5, γ: 0.01, dropout: 0.5, L_{2_a}: 0.01, L_{2_b}: 0.0005, techniques: [1, 1, 1]
Pubmed	GCN	layers: 2, lr: 0.005, hidden: 256, dropout: 0.7, L_2: 0.0005
	GAT	layers: 2, lr: 0.01, hidden: 512, dropout: 0.5, L_2: 0.0005
	APPNP	layers: 8, lr: 0.01, hidden: 64, α: 0.1, dropout: 0.5, L_2: 0.0005
	GCNII	layers: 16, lr: 0.01, hidden: 256, α_ℓ: 0.1, λ: 0.4, dropout: 0.5, $L_{2_a} = L_{2_b}$: 0.0005
	GCNIII	layers: 16, lr: 0.01, hidden: 256, α_ℓ: 0.1, λ: 0.4, γ: 0.02, dropout: 0.5, $L_{2_a} = L_{2_b}$: 0.0005, techniques: [1, 1, 1]

Table 8 summarizes the training configuration of GCNIII for full-supervised.

Table 8. The hyperparameters for Table 4.

Dataset	Model	Hyperparameters
Cora	GCNII	layers: 64, lr: 0.01, hidden: 64, α_l: 0.2, λ: 0.5, dropout: 0.5, $L_{2_a} = L_{2_b}$: 0.0001
	GCNIII	layers: 8, lr: 0.01, hidden: 64, α_l: 0.2, λ: 0, γ: 0.02, dropout: 0.5, $L_{2_a} = L_{2_b}$: 0.0001, techniques: [1, 1, 0]
Citeseer	GCNII	layers: 64, lr: 0.01, hidden: 64, α_l: 0.5, λ: 0.5, dropout: 0.5, $L_{2_a} = L_{2_b}$: 5e-6
	GCNIII	layers: 8, lr: 0.01, hidden: 128, α_l: 0.5, λ: 1, γ: 0.02, dropout: 0.5, $L_{2_a} = L_{2_b}$: 5e-6, techniques: [1, 1, 0]
Pubmed	GCNII	layers: 64, lr: 0.01, hidden: 64, α_l: 0.1, λ: 0.5, dropout: 0.5, $L_{2_a} = L_{2_b}$: 5e-6
	GCNIII	layers: 32, lr: 0.01, hidden: 64, α_l: 0.1, λ: 0.5, γ: 0.02, dropout: 0.6, $L_{2_a} = L_{2_b}$: 5e-6, techniques: [1, 1, 1]
Chameleon	GCNII	layers: 8, lr: 0.01, hidden: 64, α_l: 0.2, λ: 1.5, dropout: 0.5, $L_{2_a} = L_{2_b}$: 0.0005
	GCNIII	layers: 2, lr: 0.01, hidden: 64, α_l: 0, λ: 0, γ: 0.05, dropout: 0, $L_{2_a} = L_{2_b}$: 0.0005, techniques: [1, 0, 0]
Cornell	GCNII	layers: 16, lr: 0.01, hidden: 64, α_l: 0.5, λ: 1, dropout: 0.5, $L_{2_a} = L_{2_b}$: 0.001
	GCNIII	layers: 2, lr: 0.01, hidden: 64, α_l: 0.8, λ: 1, γ: 0.02, dropout: 0.5, $L_{2_a} = L_{2_b}$: 0.001, techniques: [1, 1, 1]
Texas	GCNII	layers: 32, lr: 0.01, hidden: 64, α_l: 0.5, λ: 1.5, dropout: 0.5, $L_{2_a} = L_{2_b}$: 0.0001
	GCNIII	layers: 2, lr: 0.01, hidden: 64, α_l: 0.5, λ: 1.5, γ: 0.05, dropout: 0.5, $L_{2_a} = L_{2_b}$: 0.0001, techniques: [1, 1, 1]
Wisconsin	GCNII	layers: 16, lr: 0.01, hidden: 64, α_l: 0.5, λ: 1, dropout: 0.5, $L_{2_a} = L_{2_b}$: 0.0005
	GCNIII	layers: 3, lr: 0.01, hidden: 64, α_l: 0.6, λ: 1, γ: 0.1, dropout: 0.8, $L_{2_a} = L_{2_b}$: 0.0005, techniques: [1, 1, 1]

Table 9 summarizes the training configuration of GCNIII for inductive learning. Following Veličković et al. (2018), we add a skip connection from the l-th layer to the $(l+1)$-th layer of GCNIII to speed up the convergence of the training process.

Table 9. The hyperparameters for Table 3.

Model	Hyperparameters
GCNII	layers: 9, lr: 0.01, hidden: 2048, α_ℓ: 0.5, λ: 1.0, dropout: 0.2, L_{2_a}: 0.0, L_{2_b}: 0.0
GCNIII	layers: 9, lr: 0.01, hidden: 2048, α_ℓ: 0.5, λ: 1.0, γ: 0.02, dropout: 0.2, L_{2_a}: 0.0, L_{2_b}: 0.0, techniques: [1, 1, 1]

It should be emphasized that we try to avoid using Dropedge in experiments, because dropedge will change the graph structure information during training, and this paper focuses on the role of node features.

C. Linear Models for Node Classification.

Yang et al. (2023) has confirmed the feasibility of MLPs (Rumelhart et al., 1986) for node classification tasks, so we wanted to explore the feasibility of linear models. Since the goal of the task is to classify nodes, we explore the capability of a linear classification model in Equation (7) with the most basic semi-supervised node classification task on Cora, Citeseer, and Pubmed Datasets. We compare the linear classification model with the 2-layer MLP and 2-layer GCN, and explore adding Batch Normalization to these models. We also evaluate the effect of *Intersect memory* technique on the linear model. The effects of Dropout are unpredictable, and in order to facilitate a more intuitive comparison between linear model, MLP and GCN, Dropout is temporarily excluded from all models. We conduct a fixed split of training/validation/testing (Yang et al., 2016) on the Cora, Citeseer, and Pubmed datasets, with 20 nodes per class for training, 500 nodes for validation and 1,000 nodes for testing. For MLP and GCN, we fix the number of hidden units to 64 on all datasets and use ReLU as the activation function. We train models using the Adam optimizer with a learning rate of 0.01 and L_2 regularization of 0.0005 for 200 epochs. We report the final training loss(float), training accuracy(%), and test accuracy(%) in Table 10, where BN represents Batch Normalization and IMLinear represents the linear classification model with *Intersect memory*.

Table 10. Evaluation of the linear classification model in the semi-supervised node classification task.

Dataset	Cora			Citeseer			Pubmed		
Model	final loss	train acc	test acc	final loss	train acc	test acc	final loss	train acc	test acc
Linear	1.2971	100.0	54.0	1.3550	100.0	54.4	0.5797	100.0	71.6
Linear(+BN)	0.0016	100.0	37.6	0.0008	100.0	39.7	0.0016	100.0	52.7
IMLinear	1.5371	95.7	72.5	1.5514	95.8	65.6	0.7704	98.3	74.2
IMLinear(+BN)	0.0074	100.0	63.4	0.0027	100.0	49.7	0.0086	100.0	60.1
MLP	0.0772	100.0	59.7	0.0923	100.0	60.3	0.0329	100.0	72.6
MLP(+BN)	0.0010	100.0	46.5	0.0009	100.0	47.6	0.0006	100.0	64.6
GCN	0.1365	100.0	80.6	0.1831	100.0	71.5	0.0669	100.0	79.8
GCN(+BN)	0.0016	100.0	76.4	0.0015	100.0	64.1	0.0009	100.0	75.2

We observe that the linear classification model can achieve 100% accuracy on the training set, but the generalization ability is significantly weak, even a little weaker than 2-layer MLP. *Intersect memory* can bring a large generalization ability to the linear classification model, while Batch Normalization can reduce the generalization ability of the models. However, the number of nodes used for training in the semi-supervised task is too small to indicate the capability of the linear models. In general, when all nodes in the dataset are used to supervise training, the training accuracy is the upper limit of the model's ability. So we repeat the experiment above, but use all the nodes for training.

Table 11. Evaluation of the linear classification model using all nodes for training.

Dateset	Cora		Citeseer		Pubmed	
Model	final loss	train acc	final loss	train acc	final loss	train acc
Linear	1.5850	61.23	1.6185	71.81	0.8238	80.69
Linear(+BN)	0.0428	100.00	0.0295	99.94	0.2859	89.80
IMLinear	1.6286	61.41	1.6430	73.28	0.9049	79.49
IMLinear(+BN)	0.0721	99.63	0.1063	97.14	0.3042	89.96
MLP	0.2764	97.16	0.3953	93.12	0.3195	88.95
MLP(+BN)	0.0031	100.00	0.0039	99.94	0.0088	99.99
GCN	0.3554	92.10	0.5628	83.62	0.3619	87.59
GCN(+BN)	0.0182	99.63	0.0672	96.93	0.0630	98.11

From Table 11, we observe that the linear models have a poor classification ability when faced with more data, much lower than MLP. But we accidentally find that Batch Normalization can improve this situation, and also improve the classification ability of MLP and GCN. Combining the results of the two groups of experiments, we basically verify that the linear models have the ability to handle the task of node classification. Batch Normalization can improve the classification ability of the models, but it will degrade the generalization ability, so it needs to be used selectively according to the actual situation.

D. Linear Transformation of Deep GCNs.

As is well known, *over-smoothing* is a major cause of the performance degradation in deep GCNs. However, DNNs without graph convolution alse face challenges in parameter training as the number of layers increases. Each layer of the classical GCN model contains a parameterized *Linear Transformation*, and we aim to explore its impact on deep GCNs. We still conduct the experiment on the semi-supervised node classification task, with the same setup in Appendix C. We compare GCN with its variant, GCN-v, which removes the linear transformation in each layer and retains only the graph convolution. It is important to note that Dropout can improve the generalization ability of GCN to some extent; however, to more clearly observe the effect of *Linear Transformation* in the comparison experiment, we avoid using this technique.

Table 12. Evaluation of *Linear Transformation* in deep GCN.

Dateset	Layer	2	3	4	5	8	16	32	64
Cora	GCN	80.5	80.3	75.4	71.8	57.4	27.0	27.2	24.2
	GCN-v	80.8	81.1	80.5	80.8	80.7	80.2	78.4	72.8
Citeseer	GCN	71.6	66.2	53.5	51.6	24.9	22.2	22.6	22.2
	GCN-v	70.8	69.3	69.5	69.2	70.0	70.5	71.1	68.9
Pubmed	GCN	80.0	78.4	75.2	74.4	63.0	44.1	40.1	42.8
	GCN-v	78.8	78.5	79.4	79.5	79.3	78.3	75.4	71.0

Table 12 reports the classification accuracy of GCN and GCN-v. It is evident that the accuracy of GCN decreases sharply as the number of layers increases, while the accuracy of GCN-v without *Linear Transformation* remains stable until the number of layers reaches 32. Thus, we conclude that parameterized *Linear Transformation*, which is difficult to optimize, is the primary cause of the poor performance of deep GCNs, whereas *over-smoothing* plays a less significant role.

E. Analysis of Node Features.

The input data for node classification typically consists of two components: the initial node features and the graph structure. Most previous studies have focused on the graph structure, while we aim to explore the impact of node features on the performance of GCNs in node classification tasks. We still follow the experimental settings in Appendix C and use a 2-layer GCN for semi-supervised learning on the Cora, CiteSeer, and PubMed datasets. We will conduct comparative experiments using the following node features: 1) Randomly generated features, which can be divided into binary discrete features and dense continuous features. 2) One-hot label encoding of the nodes, that is, the feature of the i-th node in graph $\mathcal{G}$ is a one-hot vector with the i-th position set to 1 and all other positions set to 0. The feature matrix of the entire graph $\mathcal{G}$ is the identity matrix $\mathbf{I} \in \mathbb{R}^{n \times n}$. 3) Learnable parameters, which are equivalent to the features obtained by applying a linear transformation of the same dimension to the one-hot vector features. 4) Bag-of-words representation of the nodes, which is a common initial feature for these datasets. These node features, proposed by Sen et al. (2008), is very sparse and discrete. We also consider adding a dropout layer with 0.5 rate after this feature to verify the impact of randomly dropping some features during training on the model's performance.

Table 13. Evaluation of the node features.

Dataset	Cora		Citeseer		Pubmed	
Feature	dimension	accuracy	dimension	accuracy	dimension	accuracy
Random(0-1)	1000	54.2	1000	32.6	1000	36.3
Random(dense)	1000	29.3	1000	28.3	1000	34.0
One-hot	2708	63.1	3327	33.3	19717	38.2
Learnable parameters	1000	57.4	1000	29.5	1000	33.3
Bag-of-words	1433	80.7	3703	71.5	500	79.8
Bag-of-words(dropout)	1433	82.1	3703	71.0	500	78.1

The results in Table 13 confirm that the quality of node features plays an extremely important role in the performance of GCN for node classification, which inspires us to explore constructing node features using powerful large language models(LLMs).

F. Experimental Records.

The experimental results for the semi-supervised node classification tasks are detailed in Table 14. We use the hyperparameter settings reported in Luo et al. (2024b), but do not achieve the same accuracy as in Luo et al. (2024b), and the performance of GAT even degrades with these settings.

Table 14. The results for Table 2.

Dataset	Mdoel	Results	Mean	Std
Cora	GCN	[81.9, 81.5, 81.5, 83.2, 82.9, 81.8, 81.0, 81.7, 81.8, 82.0]	81.9	0.6
	GAT	[81.4, 80.6, 80.5, 81.6, 80.0, 80.6, 80.6, 81.4, 79.8, 81.0]	80.8	0.6
	APPNP	[83.5, 83.3, 83.3, 83.3, 83.6, 83.6, 83.2, 83.3, 82.9, 82.8]	83.3	0.3
	GCNII	[84.9, 85.9, 85.2, 84.9, 85.1, 84.8, 84.7, 85.4, 85.7, 85.1]	85.2	0.4
	GCNIII	[85.6, 85.8, 84.9, 85.3, **86.1**, 84.9, 85.7, 85.6, 85.9, 85.7]	85.6	0.4
Citeseer	GCN	[71.7, 71.7, 71.7, 71.9, 72.0, 71.7, 71.6, 71.9, 71.9, 72.0]	71.8	0.1
	GAT	[69.4, 70.3, 68.3, 69.2, 68.5, 69.8, 70.1, 68.5, 70.5, 68.5]	69.3	0.8
	APPNP	[71.8, 72.0, 71.4, 71.6, 71.2, 72.0, 71.8, 72.5, 71.8, 71.5]	71.8	0.3
	GCNII	[72.9, 73.3, 73.4, 71.7, 72.2, 72.8, 71.9, 72.7, 73.3, 73.4]	72.8	0.6
	GCNIII	[73.2, 72.3, 72.9, 73.1, 73.0, 72.7, 72.3, **74.0**, 73.1, 73.5]	73.0	0.5
Pubmed	GCN	[80.0, 79.6, 79.3, 79.5, 79.3, 79.7, 79.0, 79.5, 79.3, 79.4]	79.5	0.3
	GAT	[79.3, 78.1, 77.1, 77.7, 78.6, 80.2, 78.5, 79.2, 78.3, 77.3]	78.4	0.9
	APPNP	[80.0, 79.9, 80.2, 80.4, 80.0, 80.3, 80.4, 80.1, 80.2, 79.9]	80.1	0.2
	GCNII	[79.9, 79.3, 79.3, 80.1, 79.8, 80.1, 80.6, 80.0, 79.5, 79.8]	79.8	0.4
	GCNIII	[80.0, 80.2, 80.0, **81.4**, 80.4, 80.6, 80.3, 80.6, 79.8, 80.5]	80.4	0.4

G. Out-of-Distribution Generalization of GCNII.

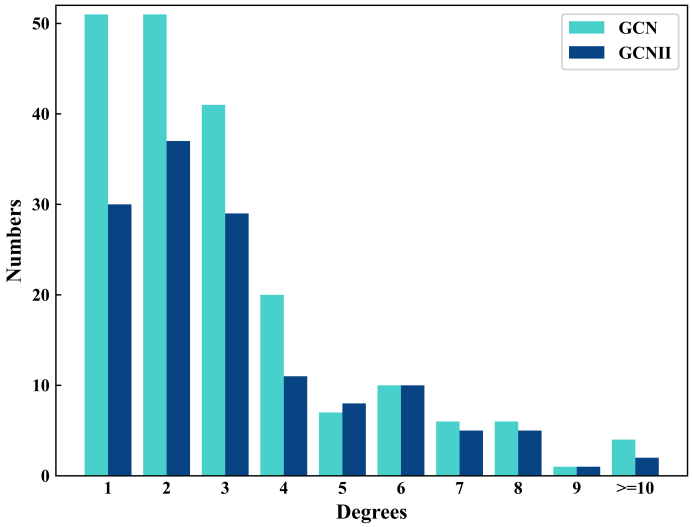

Figure 6. Degree distribution of misclassified nodes of 2-layer GCN and 64-layer GCNII on Cora.

Out-of-distribution generalization refers to the model's ability to maintain strong performance when tested on data that differs from the distribution of the training data. For graph data, isolated point pairs that are connected only to each other and isolated subgraphs that are connected only internally can be considered out-of-distribution data. From Figure 6, we can observe that the number of misclassified nodes with small degrees of GCNII is significantly reduced. Many of these nodes are the out-of-distribution data we mentioned above, and there is no path connection between them and the training nodes. No matter how deep GCN model is used, the feature information of these nodes cannot be observed during the training process. Therefore, it demonstrates that GCNII has stronger out-of-distribution generalization ability.