Probability Calibration for Graph Representation Learning

Tong Liu

München 2021

Wahrscheinlichkeitskalibrierung für das Lernen von Graphdarstellungen

Tong Liu

München 2021

Probability Calibration for Graph Representation Learning

Tong Liu

Masterarbeit
an der Fakultät für Physik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Tong Liu
aus China

Erster Supervisor: Prof. Dr. Armin Scrinzi
Zweiter Supervisor: Yushan Liu

München, den 01. August 2021

Declaration

I hereby declare that this thesis is my own work, and that I have not used any sources and aids other than those stated in the thesis.

München,
Date of submission:
Author's signature:

Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to express my biggest gratitude to Yushan Liu! She has been very nice and supportive during the whole period of the thesis. She gave me an opportunity to write a thesis from a physics student and gave me valuable guidelines and feedback. The contract is really important for me to maintain my living in Munich. Thank Dr. Marcel Hildebrandt for many times' insightful and encouraging discussions. Thank Yushan, Marcel, Dr. Mitchell Joblin, and Serghei Mogoreanu for patiently introducing the use case and answer my questions, and make me think about research from a practical perspective. Chats with Hang Li over daily lunch since last month also gave me inspiration and support. I could not thank all of you in our group enough due to a lack of words. Thanks for Prof. Dr. Armin Scrinzi, who agreed to be my internal supervisor. This is important. He also suggested me to do universal research and remember a physics perspective. During revising the thesis, the suggestions about the style and accessibility benefit the thesis. Thank Prof. Dr. Volker Tresp for providing the research environment in LMU and Siemens. Thank my family to give me 100% freedom to have my own way abroad.

Chapter 1

Introduction

1.1 Physics and graphs

What is a graph? Graphs are a ubiquitous data structure and a universal language to describe complex systems. In most cases, a graph consists of objects (i.e., nodes), along with interactions (i.e., edges) between pairs of these objects, as shown in Fig. 1.1. Rather than only considering properties of individual points, graph formalism focuses more on their relationships, which leads to its generality. The same graph formalism can be used in various areas.

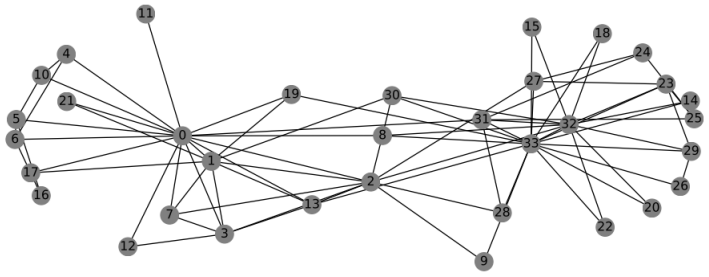


Figure 1.1: Zachary Karate Club Network represents the friendship relationships between members of a karate club. [1]

A heterogeneous graph is a graph that has multiple types of links. A knowledge graph is a special kind of heterogeneous graphs where the links are directed.
Graphs are used in various physical applications, such as predicting the properties of molecules, drug repurposing, and processing data in high-

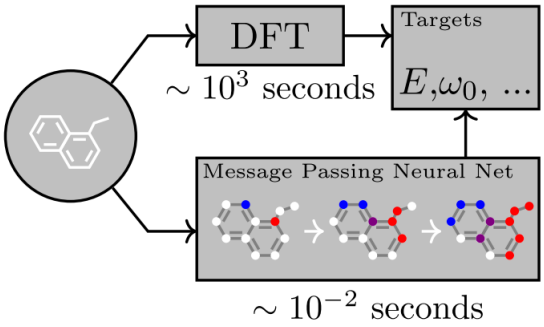


Figure 1.2: A Message Passing Neural Network predicts properties of a molecule, while DFT calculation is computationally expensive.[2]

energy physics experiments. In the following, three examples are described in more detail.

Properties of molecules. Predicting the properties of molecules is a fundamental task in physics. Density functional theory (DFT) [12] (a method to solve the time-independent Schrödinger equation) is commonly used for molecular property prediction. But it is time-consuming because it solves some big linear equations and the complexity of DFT is $O(N^3)$, where N is the number of electrons in the system.
In comparison, graph neural networks (GNNs) have been greatly used to predict the quantum mechanical properties of molecules. Researchers treat the molecule as a graph, where the nodes denote atoms and edges denote the interactions between atoms. Neural layers project each node into latent space and pass its interaction message iteratively. Then the node messages can be aggregated to represent the molecule for property prediction, as shown in Fig. 1.2. An example of the dataset is QM9 [13], where the task is to predict molecules' energetic, electronic and thermodynamic properties.

Drug repurposing. Biophysical researchers studying computational approaches rely on structured data with multi types of links, thus knowledge graphs are becoming a key instrument in biomedical knowledge discovery and modeling. Knowledge graphs are directed graph models that have different types of edges. Liu et al. [3] used knowledge graphs to do drug repurposing. It used head and tail to represent different compounds and diseases, used an edge to represent either treating between one compound

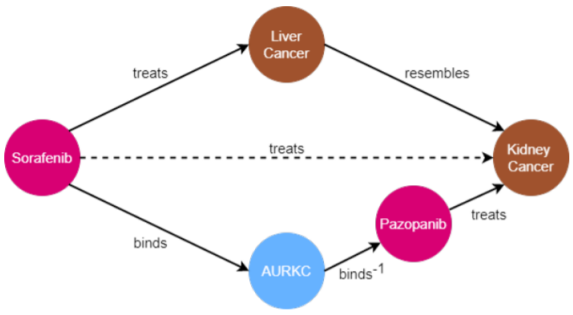


Figure 1.3: Illustration of the drug repurposing use case. [3]

and one disease, or resembling between two compounds or two diseases. For example in Fig. 1.3, the two paths that connect the chemical compound sorafenib and the disease kidney cancer can be used to predict a direct edge between the two entities.

Particle physics. The application of graph networks into high-energy physics (HEP) experiments is evolving rapidly. Physicists have made an advance to embed graph neural networks into experimental particle physics to process detector data directly [14].
At many levels, the data of HEP are sets (unordered collection) of items. Physicists consider the geometrical or physical relation between items (i.e., different detectors of different sizes using different technology to measure the trace of particles) as transformation into a graph with an adjacency matrix, to apply deep learning to learn the representation of data [15].

1.2 Motivation

In real-world decision-making systems, classification networks should not only be accurate but also should indicate when they are likely to be incorrect. As an example, consider a self-driving car that uses a deep learning model to detect obstructions where the probability of being an obstruction is important. In other words, the scores associated with the predicted class label are *calibrated* means these probabilities can reflect their ground truth correctness likelihood, e.g., if there are 100 samples each with a 0.8 probability to be of class k, then with perfect calibration there should be 80

samples that are really of class k. We will describe the mathematical and detailed definition in Chapter 2.2.1.
It has been shown that modern neural networks are mostly overconfident [16] and knowledge graph embedding (KGE) models, as well as graph neural networks (GNNs), are both poorly calibrated. Existing calibration techniques seem to be able to improve the calibration of KGE models for link prediction tasks [17] but seem to be rather ineffective when used on graph neural networks for node classification [18]. In my thesis, we mainly focus on node classification in graph neural networks. The definition and details about node classification tasks are shown in Chapter 2.1.1.
The first goal of this thesis aims at gaining a better understanding of what affects the calibration of KGE models or GNNs by investigating existing calibration techniques and their relationship to specific properties of graph representation learning models. With a better understanding, further methods could be developed to improve the calibration for graph representation learning.

1.3 Own work and outline

Own work and outline. We focus on graph representation learning which has been gaining increasing attention in the machine learning community. The calibration of graph learning models has not been sufficiently explored yet.
We elaborate on the background of graph learning and existing calibration methods in Chapter 2. Section 2.1 introduces graph representation learning, including the graph neural networks (GNNs). Section 2.2 introduces the definition and evaluation metrics of calibration, and existing benchmark calibration methods.
In Chapter 3, we do an empirical analysis of the calibration of GNNs. In Section 3.1, we investigate benchmark GNN models' calibration on citation datasets and study the effect of benchmark post-processing calibration methods. In Section 3.2 and 3.3, we analyze the relationship between calibration and properties of graph models, e.g. model capacity, aggregation, and regularization, and the influence of dataset characteristics.

Chapter 4 and Chapter 5 are concerned with the specialized machine learning community topic and are included here to complete for a comprehensive result. In Chapter 4 we evaluate the influence of knowledge distillation (KD) on calibration, since KD is thought to have a benefit of improving calibration implicitly [19, 8, 20]. In Section 4.1 and 4.2, we introduce the basic KD and a KD method in the context of graph learning. In Section 4.3, we first focus on the above-introduced methods on GNNs and further conduct a general study to analyze the influence of KD on calibration. Chapter 5 is from another view. Diving deep into GNN models, in many ways GNNs are indeed spectral methods from the graph signal processing perspective [21, 10]. The nature of the learning process of these spectral GNNs can be seen as a low-pass filtering [22]. We explore the influence of low-pass filtering on calibration in Chapter 5. With an introduction in Section 5.1 that is more of in the machine learning community, Section 5.2 is a preliminary experimental analysis.
In Chapter 6, we explore the effectiveness of a combined loss function that optimizes both accuracy and calibration. The motivation is that one loss function is hard to satisfy the minima of accuracy and probability calibration.
In Chapter 7, we propose a new calibration method that also takes the topology into account. We elaborate on the motivation, methods and results in Sections 7.1, 7.2 and 7.3.
Chapter 8 is an application to a real-world use case. Section 8.1 and 8.2 are the background and the analysis of calibration results. Then we set a higher demand for calibration methods: to learn compatibility information and propose our metrics to evaluate it. In Section 8.4 we further explore the influence of different settings on calibration.
Finally a summary of results of each part above is in Chapter 9.

Main contributions of the thesis are summarized:

1. We investigate benchmark GNN models' calibration, apply existing post-processing calibration methods and explore the influence of properties of graph models, e.g., model capacity, aggregation and regularization. (Chapter 3)

2. We show the effectiveness of replacing the original loss with a combined loss function that optimizes both accuracy and calibration. (Chapter 6)

3. We propose a novel topology-aware calibration method that outperforms more than half of state-of-the-art calibration methods. (Chapter 7)

4. In application to a real use case, calibration methods could not only calibrate the model but also help learn some compatibility information. (Chapter 8)

Chapter 2

Fundamentals

In this chapter, we give the basic knowledge about this thesis. It mainly covers three parts: graph representation learning (Chapter 2.1), probability calibration (Chapter 2.2) and existing calibration work (Chapter 2.3).

2.1 Graph representation learning

2.1.1 Machine learning on graphs

Machine learning tasks on graphs. Graphs are a ubiquitous data structure and a universal language to describe complex systems, like in Fig. 1.1. More formally, a graph is represented as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the edges between the nodes. We denote an edge in $\mathcal{V}$ going from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ as $(u, v) \in \mathcal{V}$. $A \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ is an adjacency matrix, where $A[u, v] = 1$ if $(u, v) \in \mathcal{V}$ and $A[u, v] = 0$ otherwise. The edge between node u and v is undirected if the edges from both sides are equivalent, $(u, v) \in \mathcal{V} \leftrightarrow (v, u) \in \mathcal{V}$. If a graph contains only undirected edges (so called undirected graph), then A is symmetric. If a graph is directed graph (the direction also matters), A is not necessarily symmetric. For attributed graphs (Each node is associated a D-dimensional feature vector. Feature is a node-level information in a graph, e.g., a profile figure associated to a user in a social network.), we denote the feature matrix with $X \in \mathbb{R}^{|\mathcal{V}|\times D}$. Degree matrix D of an undirected graph is a diagonal matrix which describes the degree of each vertex (denotes the number of neighbors of the node): $D \in \mathbb{R}^{|\mathcal{V}|\times\mathcal{V}}$, $D_{ii} = deg(i)$ and $D_{ij(i\neq j)=0}$. The graph Laplacian

is defined as $L = D - A \in \mathbb{R}^{|\mathcal{V}|\times\mathcal{V}}$, where D and A are degree matrix and adjacency matrix in the graph.
In some cases, we extend the edge notation in a graph to include multiple types. These graphs are called multi-relational graphs. A knowledge graph is a directed multi-relational graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$, where $\mathcal{R}$ is the set of relation, with edges denoted as tuples $e = (u, \tau, v)$ indicating there is a link of type τ between head u and tail v.
The usual categories of supervised and unsupervised tasks are not the most useful category when it comes to graphs. Usually, there are several following tasks on graphs. Let $\mathcal{V}_{train} \subseteq \mathcal{V}$ and $\mathcal{E}_{train} \subseteq \mathcal{E}$ denote nodes with true labels available in training and edges available in training respectively. **Node classification** is to predict the label y_u for node u, when we are only given the true labels on the training set of the nodes $\mathcal{V}_{train}$. **Link prediction** is to predict the relationship of edges $\mathcal{E}\backslash\mathcal{E}_{train}$, when we are given node set $\mathcal{V}$ and an incomplete set of edges $\mathcal{E}_{train}$. **Community detection** is analogues of unsupervised clustering, where the task is to cluster the graph into different communities where nodes are more likely to form edges with nodes in the same community. **Graph classification and clustering** view each graph as an i.i.d. datapoint with a label, and the goal is to learn a mapping from datapoints to labels.

2.1.2 Node embeddings

The key process in machine learning on graphs is to find a way to incorporate information of graphs into a machine learning model. But the challenge is that there is no straightforward way to use the graph's information [23].
Following Ref. [24], learning *node embeddings* aims at learning some low-dimensional vectors, so that geometric relationships in the embedding space reflect the structure of the original graph, as shown in Fig. 2.1. Node $u \in \mathcal{V}$ and node $v \in \mathcal{V}$ are neighbors in the original graph. In machine learning, encoders denote one kind of models that learn the representation of data, such that the important information of the input data is retained. An encoding function $ENC : \mathcal{V} \to \mathbb{R}^d$ is such a similar function. After finishing optimizing ENC, the embedding vectors for node u and v, $z_u = ENC(u) \in \mathbb{R}^d$, and $z_v = ENC(v) \in \mathbb{R}^d$ can also represent the neighborhood

information of u and v.

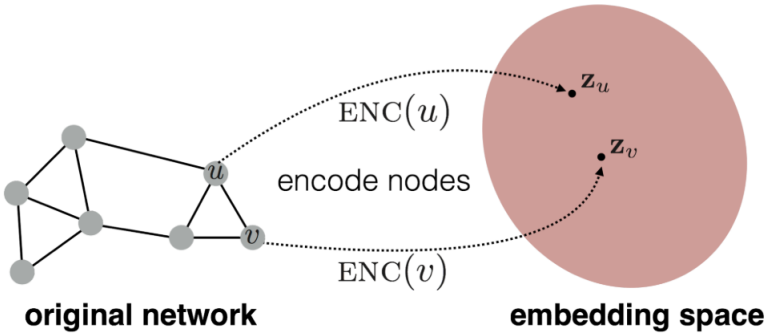


Figure 2.1: Illustration of node embedding problem. [1]

2.1.3 Graph neural networks

Graph neural network (GNN) formalism is one kind of encoder models that use neural networks to encode nodes in the original graph. It generates representations of nodes that depend on both the graph structure and the feature information. We need to define the learning framework for GNN, which is called *neural message passing*. Neural message passing includes *neighborhood aggregation* and *updating*.

2.1.3.1 Neural message passing

Framework of message passing. *Neural message passing* means messages are exchanged with neighbors and updated to each node. With GNN-formalism encoder models, we can generate a hidden embedding $h_u \in \mathbb{R}^d$ for node u given an input graph G and node feature matrix X. The generating method is that during each iteration of GNN, for node u, the information is first aggregated from u's neighbors $\mathcal{N}(u)$, then updated to node u. Therefore, the message passing includes two processes: *aggregation* from neighbors and *updating* itself. Following Ref. [24], the framework can be represented as:

$$\begin{aligned} h_u^{(k+1)} &= UPDATE^{(k)}(h_u^{(k)}, AGGREGATE^{(k)}(h_v^{(k)}, \forall v \in \mathcal{N}(u))) \\ &= UPDATE^{(k)}(h_u^{(k)}, m_{\mathcal{N}(u)}^{(k)}), \end{aligned} \tag{2.1}$$

where UPDATE and AGGREGATE are two arbitrary differentiable functions that represent updating and aggregation respectively. $h_u^{(k)} \in \mathbb{R}^{d^{(k)}}$ is a hidden embedding of node u for k-th layer, which is updated according to the information aggregated from $u's$ neighborhood $\mathcal{N}(u)$. $m_{\mathcal{N}(u)}^{(k)}$ is the message aggregated and prepared to be updated to $(k+1)$-th layer hidden embedding $h_u^{(k+1)}$. Usually the initial embedding $h_u^{(0)}$ is directly set to input features x_u, i.e., $h_u^{(0)} = x_u$. The output of final layer after K iterations, i.e., $z_u = h_u^{(K)}$, can be the model output.

The basic GNN. Following Ref. [24], a simple concrete formalism to UPDATE and AGGREGATE functions can be seen as following. Or to say, the basic GNN message passing is defined as:

$$h_u^{(k+1)} = \sigma(W_{self}^{(k+1)} h_u^{(k)} + W_{neighbor}^{(k+1)} \sum_{v \in \mathcal{N}(u)} h_v^{(k)} + b^{(k+1)}), \tag{2.2}$$

where $W_{self}^{(k+1)}$, $W_{neighbor}^{(k+1)} \in \mathbb{R}^{d^{(k+1)} \times d^k}$ are parameter matrices, σ is non-linearity and $b^{(k+1)} \in \mathbb{R}^{d^{(k+1)}}$ is a bias term. Here we use a SUM function as AGGREGATE, i.e. $m_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} h_v$, and a perceptron (a weight matrix times input matrix, then input to an non-linear function) as UPDATE, i.e. $h_u^{'} = \sigma(W_{self} h_u + W_{neighbor} m_{\mathcal{N}(u)})$.

Message passing with self-loops. Following Ref. [24], as a simplification of neural message passing approach, it is common to write together the aggregation from neighbors and the node itself, or to say, *message passing with self-loops*. More precisely,

$$h^{(k+1)} = AGGREGATE(h^{(k)}, \forall v \in \mathcal{N}(u) \cup u), \tag{2.3}$$

We can write it in the matrix form for all nodes corresponding to Equation 2.2,

$$H^{(k+1)} = \sigma((A+I) H^{(k)} W^{(k+1)}), \tag{2.4}$$

where $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ is an adjacency matrix, I is an identity matrix. $H^{k+1} \in \mathbb{R}^{|\mathcal{V}| \times d^{(k+1)}}$ and $W^{k+1} \in \mathbb{R}^{d^{(k)} \times d^{(k+1)}}$ denote the $(k+1)$-th layer's embedding matrix and weight matrix. Actually this can also be derived from first-order expansion of N-polynomials of adjacency matrix due to translation equivariance (see Eq. 5.6 and 5.14).

2.1.3.2 Neighborhood aggregation

Neighborhood normalization. Equation 2.2 defines the most basic neighborhood aggregation operation. But we cannot tolerate sometimes one neighbor's embedding could be many times larger than the value of one another. Therefore we need to normalize aggregation among all neighbors. The simplest approach is to take an averaged [23]:

$$m_{N(u)} = \frac{\sum_{v \in \mathcal{N}(u)} h_v}{|\mathcal{N}(u)|}. \tag{2.5}$$

Or use symmetric normalization (also normalize among the neighbor's neighbors) like in Graph Convolutional Network (GCN) [21]:

$$m_{N(u)} = \sum_{v \in \mathcal{N}(u)} \frac{h_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}. \tag{2.6}$$

With the message passing with self-loops and symmetric normaliztion, we can write the message passing function the same as GCN's:

$$h_u^{(k)} = \sigma(W^{(k)} \sum_{v \in \mathcal{N}(u) \cup u} \frac{h_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}). \tag{2.7}$$

GCN can also be equivalently derived from a graph signal processing perspective (see Chapter 5).

Neighborhood attention. A natural ideas to extend GCN is to consider weight on each edge, or to say, to apply attention to edges: more important neighbors take higher attention weights. The first paper to introduce this idea is Graph Attention Network (GAT) [25], which uses attention weights to define a weighted sum of neighbors:

$$m_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} h_v, \tag{2.8}$$

where $\alpha_{u,v}$ defines the attention from v to u. In this way, we aggregate information from neighbors of node v to node u with attention. The attention $\alpha_{u,v}$ is computed as:

$$\alpha_{u,v} = \frac{exp(a^T[Wh_u \bigoplus Wh_v])}{\sum_{m \in \mathcal{N}(u)} exp(a^T[Wh_u \bigoplus Wh_m])}, \tag{2.9}$$

where a is a trainable vector, W is the matrix in message passing, and $\bigoplus$ is the concatenation operation.

2.1.3.3 Updating in GNNs

Over-smoothing. The most common-used UPDATE operator is as stated in Eq. 2.2. However, one common issue when updating information in GNNs is over-smoothing [26]. It means after several iterations of message passing, the representations for all nodes start to become similar and indistinguishable. This makes it hard to build deeper layer GNN models. Section 5.1.4 gives a more theoretical perspective to understand over-smoothing from signal processing.
More precisely, the degree of over-smoothing can be formalized by defining the influence of node $u's$ input features on the final layer embedding of the other nodes as [26]:

$$I_K(u,v) = \mathbf{1}^T(\frac{\partial h_v^{(K)}}{\partial h_u^{(0)}})\mathbf{1}, \tag{2.10}$$

where $\mathbf{1}$ is a vector of all ones. This Jacobian matrix measures how much the initial embedding of node u influences the final embedding of v.
The above equation defines the influence of node u's input features on node v's final layer embedding $I_K(u,v)$. The following theorem states that $I_K(u,v)$ is proportional to the probability of reaching node v from node u on a K-step random walk (a successive random-step walk).
Theorem 2.1. [26] *For any GNN model using a self-loop update approach and an aggregation function of the form*

$$AGGREGATE(h_v, \forall v \in \mathcal{N}(u) \cup u) = \frac{1}{f(|\mathcal{N}(u) \cup u|)} \sum_{v \in \mathcal{N}(u) \cup u} h_v, \tag{2.11}$$

where f is a differentiable normalization function, we have that

$$I_K(u,v) \propto p_K(u|v), \tag{2.12}$$

where $p_K(u|v)$ denotes the probability of visiting node v on a length-K random walk starting from node u.

This theorem can also be extended to the basic GNN update without self-loop, like equation 2.2, as long as $||W_{self}^{(k)}|| < ||W_{neigh}^{(k)}||$ at each k. It states that for a K-layer GNN model, the influence of node u and node v is proportional to the probability of reaching node v on a K-step random walk starting from node u. Therefore, when K is large, the random walk range is large, the probability to reach one particular node is quite small, then the influence is small. In the limit situation $K \to \infty$, the probability to reach any node approaches to a constant, therefore the influence between any two nodes approaches the same, or to say, the local neighborhood information is lost.

2.1.3.4 Methods

In this section, we will give a list to GNNs we use in our thesis. Most methods are described before and later.

Graph Convolutional Networks (GCN)[21]. The vanilla GCN is one of the most basic GNN methods. Its spatial formalism is introduced in Section 2.1.3. In a spatial perspective, a GCN layer is defined as:

$$H^k = \sigma(\tilde{A}H^{k-1}W^k), \tag{2.13}$$

where W^k is a learnable parametric matrix.

It is also a spectral method. It can be equivalently derived from the first-order approximation of $N - th$ polynomial function of Laplacian L of the graph, as introduced in Section 5.1.5.

Graph Attention Network (GAT)[25]. Rather than only aggregating from equivalent edges from neighbors, GAT adopts self-attention, to give different weights to neighbors' edges A GAT layer is defined as:

$$H^k = \sigma(\alpha\tilde{A}H^{k-1}W^k), \tag{2.14}$$

where $\alpha \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, $\alpha_{u,v}$ defines the attention from v to u as shown in Section 2.1.3.

Simple Graph Convolution (SGC)[10]. SGC removes the non-linearity σ between each layer in GCN, resulting in not much performance decrease but huge running speed increase. A K-th layer SGC layer represented as:

$$H = \sigma(\tilde{A}^K XW), \tag{2.15}$$

where $W = W^1W^2...W^K$ is a reparameterized weight matrix and $\tilde{A} = A+I$ being the adjacency matrix with self-loops.

Graph filter neural network (gfNN)[22]. gfNN can be viewed as a simple perceptron with weight W_2 after SGC, in order to improve the expression ability. Similar to Eq. 2.15, a gfNN can be represented as:

$$H^K = \sigma(\sigma(\tilde{A}^K X W_1)W_2). \tag{2.16}$$

Approximate personalized propagation (APPNP) [27]. First, Klicpera et al. introduced Personalized Propagation of Neural Predictions (PPNP), an aggregation method based on Personalized PageRank (PPR), which is defined as $\pi_{PPR} = \alpha(I-(1-\alpha)\hat{\tilde{A}})^{-1}$, where $\alpha \in (0,1)$ is a hyperparameter, I being identity matrix, and $\hat{\tilde{A}} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ being symmetric normalized adjacency matrix with self-loop as defined in Chapter 5. The influence of node i on node j is proportional to the element (x, y) in the PPR matrix π_{PPR}. Then the PPNP method can be written as:

$$\begin{aligned} Z^{(0)} &= MLP(X) \\ Z_{PPNP} &= softmax(\pi_{PPR}Z^{(0)}), \end{aligned} \tag{2.17}$$

where X is the node features. PPNP consists of a MLP for feature transformation $Z^{(0)}$ and PPR layer for feature aggregation Z_{PPNP}.

Suppose $\pi_{PPR} \in \mathbb{R}^{n\times n}$, the computational complexity and memory of π_{PPR} is $O(n^2)$, which is not effective in training and inference [27]. Instead, Klicpera et al. introduced APPNP, an approximation of PPNP, which changes aggreagtion of Eq. 2.17 to an iterative way as:

$$\begin{aligned} Z^{(k+1)} &= (1-\alpha)\hat{\tilde{A}}Z^{(k)} + \alpha Z^{(0)} \\ Z_{APPNP} &= softmax(Z^{(K)}). \end{aligned} \tag{2.18}$$

2.1.3.5 Tasks focused in the thesis

In this thesis, node classification is the main task we focus on. Node classification is perhaps the most popular machine learning task on graphs. Definition is introduced in section 2.1.1. Examples of node classification include classification of the proteins [23] (where nodes represent proteins and

edges represent interaction between proteins) , classification of the topic of a paper based on the citation graph [21] [28] (where nodes represent papers and edges represent citation between papers), classification of the types of goods co-purchased in the co-purchased network, in Amazon-computers and Amazon-photos [28] (where nodes represent goods and edges represent items are frequently co-purchased), classification of the users' information, e.g. religions, sex and home country, in Facebook's social networks [29] (where nodes represent users and edges represent their friendship or relationship), and so on.

2.1.3.6 Semi-supervised setting

In the usual supervised setting, unlabeled test samples are completely unobserved during training. In graph learning, we usually use *semi-supervised setting* [**?**]. This term means we usually have access to the whole graph, including the whole structure and every node features. The missing part is the labels of test nodes. The structure information of test nodes and features are still input to the model.

2.2 Probability calibration

2.2.1 Model calibration

Only accuracy is not enough for many real-world decision-making systems. We need to know which predictions we should trust and which not. For models with softmax scores, we need to know if these scores can represent the real probabilities correctly. This leads to the concept of *calibration*. In a perfect calibrated model, *the softmax output should match the real frequency that the prediction is correct.*

Following Ref. [16], more formally, for a given model $f : \mathbb{R}^d \to [0, 1]^K$ with its input $X \in \mathbb{R}^{N \times d}$ (N is the number of samples) and ground-truth label $Y \in \{1, ..., K\}$ (K is the number of classes). For a given sample i, let X_i be the input for sample i and $f(X_i)$ be the output probability scores. Denote

y_i as the ground-truth label for sample i. The predicted label $\hat{y}_i$:

$$\hat{y}_i = \hat{y}(X_i) = \underset{k\in\{1,\dots,K\}}{\arg\max}[f(X_i)]_k, \tag{2.19}$$

and its predicted probability:

$$\hat{p}_i = \hat{p}(X_i) = \max_{k\in\{1,\dots,K\}}[f(X_i)]_k. \tag{2.20}$$

The perfect calibration is defined as:

$$\mathbb{P}(\hat{y}_i = y_i|\hat{p}_i = p) = p, \quad \forall p \in [0,1] \tag{2.21}$$

An simple example of perfect calibration is that if there are 100 samples each with a 0.8 probability to be of class k, then with perfect calibration there should be 80 samples that are really of class k.

2.2.2 Evaluation of calibration

There are some methods and metrics to evaluate the calibration.

2.2.2.1 Reliability diagrams

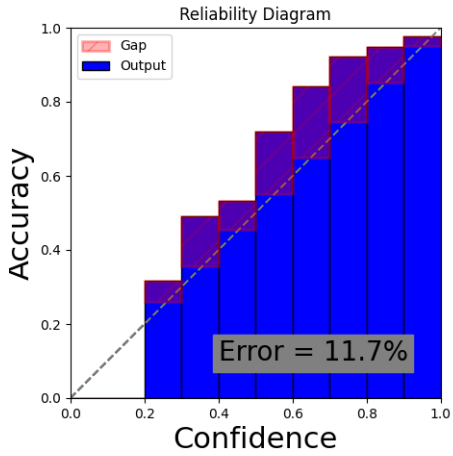


Figure 2.2: Reliability diagrams for GCN on Pubmed. Error here denotes ECE in Eq. 2.22.

Reliability diagrams [30, 31] is a visual representation of model calibration. It plots the accuracy against confidence (the biggest-class probability), as shown in Fig. 2.2. To estimate the expected accuracy, the predictions of the samples are grouped into M interval bins, according to their confidence values. For each bin B_m ($1 \leq m \leq M$), the accuracy is:

$$acc(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbb{1}[y_i = \hat{y}_i], \tag{2.22}$$

while the average confidence is:

$$conf(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i, \tag{2.23}$$

where $|B_m|$ is the number of samples in that bin, $\mathbb{1}$ is the indicator function, y_i, $\hat{y}_i$ and $\hat{p}_i$ are the true label, predicted label and predicted probability for sample i in the bin.
The perfect calibration is defined as $acc(B_m) = conf(B_m)$ for each $m \in \{1, ..., M\}$. If the output is generally above the diagonal line (as Fig. 2.2), the model outputs confidence lower than accuracy, or to say, the model or the probability is *underconfident*. If the output is generally below the diagonal line, the model or the probability is *overconfident*.

2.2.2.2 Calibration error

Expected Calibration Error (ECE) [32, 16] is a scalar which is the average of the gaps in the reliability diagram, weighted by the samples number in each bin (Error in Fig. 2.2):

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{N} |acc(B_m) - conf(B_m)|, \tag{2.24}$$

where M is the number of bins and N is the total number of samples.
$ECE_{\geq 50}$[18] is an ECE only computed for samples with confidence of higher than 50%. Since in some cases, models make mostly random predictions (usually with less than 50% confidences), which also leads to a relatively small ECE but not reliable calibration.

Maximum Expected Calibration Error (Maximum-ECE) [32], in comparison, summaries the largest deviation between confidence and accuracy in each bin B_m:

$$Maximuml\text{-}ECE = \max_{k\in\{1,\dots,M\}} |acc(B_m) - conf(B_m)|. \tag{2.25}$$

Kolmogorov-Smirnov calibration error (KS-error) [33]. ECE relies on the number of the chosen binning scheme. Different choices of number of bins lead to different ECE [34]. KS-error is introduced to overcome this shortcoming as KS is a binning-free metric by the statistics of cumulative distributions.
KS-error computes the maximum difference between cumulative accuracy and cumulative confidence. For $t \in [0, 1]$,

$$\begin{aligned} KS &= \max_t |acc(t) - conf(t)| \\ acc(t) &= \frac{1}{N}\sum_{i:\hat{p}_i\le t} \mathbb{1}[y_i = \hat{y}_i], \quad conf(t) = \frac{1}{N}\sum_{\hat{p}_i\le t} \hat{p}_i, \end{aligned} \tag{2.26}$$

where acc and $conf$ are accuracy and confidence defined in KS-error, N being the number of samples.

Marginal expected calibration error (Marginal-ECE)[34]. Marginal-ECE compares averaged probability of each class in each bin with the proportion of samples of that class in that bin:

$$Marginal\text{-}ECE = \sum_{k=1}^{K} w_k \sum_{m=1}^{M} \frac{|B_m|}{N} \left| \frac{1}{|B_m|}\sum_{i\in B_m} \mathbb{1}[k = y_i] - \frac{1}{|B_m|}\sum_{i\in B_m} f(\mathbf{X}_i)_k \right| ,$$

Although marginal-ECE is intuitively defined to make sense, it is also considered not to represent the marginal calibration very well and even collapse in some cases, e.g. in a balanced-class dataset, if we use a very high temperature to scale down all scores to about $1/k$, then almost all scores would fall into the same bin that contains $1/k$, which is equivalent to only one bin exists. In that bin, the balanced-class dataset has the same proportion of samples for each class, therefore $marginal\text{-}ECE = 0$ [35].

Negative Log-Likelihood (NLL) [36], is a loss function also known as *cross-entropy loss* in multi-class classification. NLL is defined as:

$$NLL = -\sum_{i=1}^{N} y_i \log \hat{p}_i, \tag{2.27}$$

NLL is minimal if and only if the prediction probability $\hat{p}_i$ recovers true label y_i, therefore, training with NLL loss may cause the probability $\hat{p}_i$ to be close to 1.

Brier score [37], is defined as the squared error of the predicted probability $\hat{p}_i$ and the ground truth label y_i. For classification problems, Brier score can be decomposed into a loss for accuracy and a loss for calibration[1], therefore Brier score could also represent the calibration. Brier score is defined as:

$$Brier = \frac{1}{N}\sum_{i=1}^{N}(\hat{p}_i - y_i)^2. \tag{2.28}$$

2.2.3 Theorems related to calibration

Rahimi et al. [38] gave a theoretical description about calibration. It is worthy of stating exactly.

Preliminaries[2]. Let (X, K) be a pair of joint random variables, where X takes values in domain D_X and K takes values in the class set $\kappa = \{1, ..., n\}$. n is the number of labels. Let a function $f : D_X \to D_Z = \mathbb{R}^m$ denote our NN, where $Z = f(X)$ is a random variable denoting the output. Let q denote the terminate function, or activation after the model f usually in the task of classification, e.g. q being the softmax function. $q : \mathbb{R}^m \to \mathbb{R}^n$, where $q(z) = (q_1(z), ...q_n(z))$. Here since $q_k(z)$ satisfies some condition, e.g. $\sum_{k=1}^{n} q_k(z) = 1$, we denote the set of $q_k(z)$ as the probability simplex Δ^{n-1}. Let $q \circ f$ denote the NN with activation, $q \circ f : D_X \to \Delta^{n-1}$. Given two values z and k sampled from random variables X and K (denoted as $x \sim X$

[1] We do not show here. For details people can read [37].

[2] Note, the notation here is different from the notation in the other parts of the thesis.

and $k \sim K$), let $P(k|z)$ denote the probability of K takes k and Z takes z, $P(K = k|Z = z)$.
Given a pair data $(x, k) \in D_X \times \kappa$, our expected negative log likelihood loss given the random variable (X, K) is: $L(q \circ f, X, K) = E_{(x,k)\sim(X,K)} L(q \circ f, x, k) = -E_{(x,k)\sim(X,K)} log(q_k(f(x)))$. But usually we cannot know the real distribution of random variables (X, K), therefore our expected negative log loss is approximated by empirical loss:

$$L(q \circ f, D) \approx -E_{(x,k)\sim D} log(q_k(f(x))) = -\sum_{i=1}^{N} log(q_{k_i}(f(x_i))), \tag{2.29}$$

where we denote samples of data pairs sampled from the distribution of (X, K) as $D = \{(x_i, k_i)_{i=1}^N\}$.

Calibration. Here we state a theorem of calibration. Theorem 2.2 is a combined theorem of Theorem 3.1, 3.2 and 3.3 in [38]. It gives different conditions to be calibrated for a model, a strong version, a weak version, and a third equivalent version. Proof of Theorem 2.2 can be seen in Appendix B.

Theorem 2.2. [38] *Consider joint random variables (X, Z, K), taking values from $D_X, \mathbb{R}^m$ and κ respectively. Let $f : D_X \to \Delta^{n-1}$ be our NN, $L_1(f, X, K) = -E_{(x,k)\sim(X,K)} log(f_k(x))$ be the first loss given our model f and input X, $L_2(q, Z, K) = -E_{(z,k)\sim(Z,K)} log(q_k(z)) = -E_{(x,k)\sim(X,K)} log(q_k(f(x)))$ be the second loss given our output Z, or equivalently given f and X.*

A strong version: if

$$f = \underset{f':D_X\to\Delta^{n-1}}{argmin}\ L_1(f', X, K), \tag{2.30}$$

then it is perfect calibrated with respect to f, i.e. $P(k|x) = f_k(x)$.

A weak version: if

$$\underset{g:\mathbb{R}^m\to\mathbb{R}^m}{argmin} L_2(q \circ g, Z, K) = id, \tag{2.31}$$

where id is the identify function, $g : \mathbb{R}^m \to \mathbb{R}^m$ is a modification, e.g. a post-processing calibration function, then it is perfect calibrated with respect to g, i.e. $P(k|z) = q_k(z)$.

An equivalently weak version: if

$$\underset{g:\mathbb{R}^m \to \mathbb{R}^m}{argmin}\, L_2(q \circ g \circ f, X, K) = id, \tag{2.32}$$

then it is perfect calibrated with respect to g, i.e. $P(k|z) = q_k(z)$*, or to say,* f *is perfect calibrated.*

The strong version of this theorem requires $f : D_X \to \Delta^{n-1}$ to be exactly minimized w.q.t. our final probability output $q(z) \in \Delta^{n-1}$, but it is usually impossible, even using Bayesian method [39]. Usually f is optimized w.q.t. logits $z \in \mathbb{R}^m$, therefore a weak version of this theorem appears. It does not require f to be optimal. It states that f is optimal to calibration if replacing f by $g \circ f$, i.e. using a post-processing calibration method g, can not decrease the L_2 loss.

2.3 Related work for calibration

2.3.1 Post-processing calibration methods

Post-processing calibration methods produce calibrated logits (logits denote raw outputs without softmax) (or calibrated probabilities) with input model logits (or probabilities). Usually, post-processing methods are trained in the validation set and tested in the test set. Several early post-processing approaches were not originally formulated for deep learning, such as Platt Scaling[40], Histogram Binning[41], Isotonic Regression[42], and Bayesian Binning into Quantiles[32]. After the calibration problem got an attention by deep learning researchers since Guo et al. proposed it [16], post-processing methods are rapidly developed in the context of deep learning.

2.3.1.1 Binary classification

In the binary setting, i.e., $Y = \{0, 1\}$, for a sample i, let z_i be the model's non-probabilistic output, or logit, $\hat{p}_i = \sigma(z_i)$ be the model's predicted probability (here we assume the binary model only outputs the positive class score), where σ is the sigmoid function. The goal is to produce the calibrated probability $\hat{q}_i$ based on $y_i, \hat{p}_i$ and z_i.

Histogram binning [41] first divides predicted probability $\hat{p}_i$ into M bins, then learns a score in each bin to represent calibrated probabilities. More precisely, with bins $B_1, ..., B_M$, and corresponding bin boundaries $0 = a_1 \leq ... \leq a_{M+1} = 1$, each bin B_m belongs to the interval $(a_m, a_{m+1}]$ $(m = 1, ..., M)$. The calibrated score $\hat{q}_m$ in B_m is learnt by:

$$\underset{\hat{q}_1,...\hat{q}_M}{min} \sum_{m=1}^{M} \sum_{i \in B_m} \mathbb{1}(a_m \leq \hat{p}_i < a_{m+1})(\hat{q}_m - y_i)^2, \tag{2.33}$$

For predicted probabilities belonging to $(a_m, a_{m+1}]$, histogram binning learns an averaged accuracy score $\hat{q}_m$, by minimizing the sum squared error of $\hat{q}_m$ to predicted labels y_i.

Isotonic regression [42] learns a piece-wise constant function g to transform predicted probability, $\hat{q}_i = g(\hat{p}_i)$. It's an equivalent generalization to histogram binning with bin boundaries and calibrated scores jointly optimized:

$$\begin{aligned} &\underset{M,g,a_i}{min} \sum_{m=1}^{M} \sum_{i=1}^{n} \mathbb{1}(a_m \leq \hat{p}_i < a_{m+1})(g(\hat{p}_i) - y_i)^2, \\ \textit{subject to} \quad &0 = a_1 \leq ... \leq a_{M+1} = 1, \\ &g(\hat{p}_1) \leq ... \leq g(\hat{p}_M) = 1. \end{aligned} \tag{2.34}$$

Platt scaling [40] learns scalar parameters $a, b \in \mathbb{R}$ in a logistic regression model to transform logit z_i into calibrated probability $\hat{q}_i = \sigma(az_i + b)$. a and b are optimized in validation setting with NLL loss. All samples share the same parameters a and b.

2.3.1.2 Multi-class classification

For multi-classification problems, we use softmax function σ to get predicted probability $\hat{p}_i = \underset{k}{max}\sigma(z_i)_k$. The goal is to produce a calibrated probability $\hat{q}_i$ based on true label y_i, $\hat{p}_i$ and logits z_i.

Extension of binning methods. [42] By treating the multi classification problem as K one-versus-all problems, histogram binning and isotonic re-

gression can be extended to multiclass setting. For each $k = 1, ..., K$, we have binary a classification problem of true label being 1 if $y_i = k$. Using K seperate models, we would get a unnormalized K-length probability vector, $[\hat{q}_i^{(1)}, ..., \hat{q}_i^{(K)}]$, where $\hat{q}_i^{(k)}$ is calibrated probability for the class k. After normalization, the calibrated probability vector $\hat{q}_i = \max_k \hat{q}_i^{(k)} / \sum_{k=1}^{K} \hat{q}_i^{(k)}$.

Matrix and vector scaling[16]. This is extended from Platt scaling in binary classification problem, by learning a matrix $W \in \mathbb{R}^{K \times K}$ and a vector $b \in \mathbb{R}^K$ instead of two scalars:

$$\begin{aligned} \hat{q}_i &= \max_k \sigma_{SM}(W z_i + b)^{(k)} \\ \hat{y}_i' &= \arg\max_k \sigma_{SM}(W z_i + b)^{(k)}, \end{aligned} \tag{2.35}$$

where matrix scaling becomes vector scaling when matrix W is restricted to be a diagonal matrix.

Temperature Scaling [16] uses a positive scalar T to scale the logit vector z. The calibrated probability vector is:

$$\hat{q}(z) = \sigma(\frac{z}{T}) = \frac{exp(\frac{z}{T})}{\sum_{i=1}^{K} exp(\frac{z_i}{T})}, \tag{2.36}$$

where σ is the softmax function. The parameter T is optimized with respect to NLL on the validation set. For $T > 1$, it "softens" the logits, making the model change from overconfident to more calibrated. For $T < 1$, it "hardens" the logits, making the model change from underconfident to more calibrated.
Since T does not change the ranking of probabilities of different classes, it does not affect the model's accuracy. Guo et al. [16] showed that temperature scaling works better than several above post-processing methods in computer vision context [16].

2.3.2 Built-in calibration methods

Besides post-processing calibration methods, there are also built-in calibration methods. These methods modify the learning algorithm or training

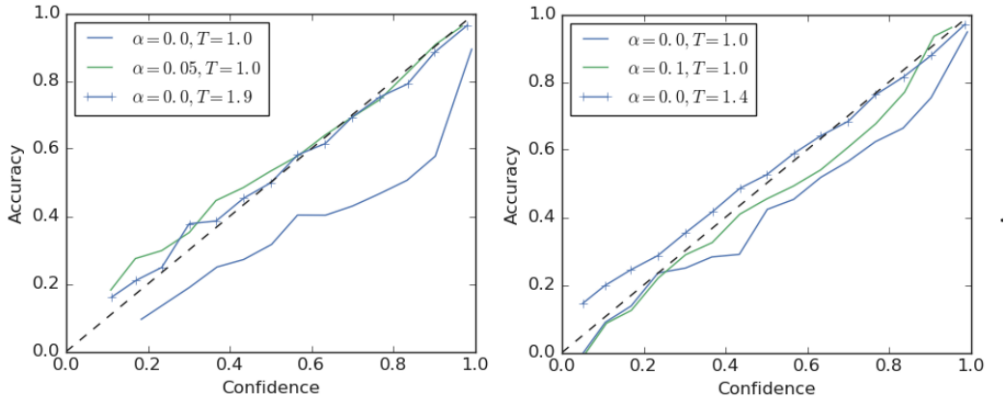


Figure 2.3: Reliability diagram for two models on two datasets. Left plot: ResNet-56 [4] on CIFAR-100 [5]. Right plot: Inception-v4 [6] on ImageNet [7][4]. Figure is from Muller et al.[8].

procedure[43]. Some methods were originally designed for regularization (denotes methods preventing model to be overfitting) and then found to be helpful to calibration [44, 45]. Some methods modify the croess-entropy loss function [46]. Here we introduce a modifying-loss-function method as an example. This method also relates to knowledge distillation[47] in Chapter 5.

.

Label smoothing (LS) [48, 8] was proposed to improve the performance of classification [48], then was shown by Muller et al. [8] to be able to help calibration implicitly. Rather than minimizing cross-entropy between the true label y_i and network output p_i as the cross-entropy loss $L_{CE}(y,p) = -\sum_{i=1}^{K} y_i log(p_i)$, LS minimizes the cross-entropy between modified label y_i^{LS} (smoothed label) and networks' output p_i as

$$L_{LS}(y,p) = -\sum_{i=1}^{K} y_i^{LS} log(p_i), \tag{2.37}$$

where y_i^{LS} is the smoothed label, defined as $y_i^{LS} = y_i(1-\alpha)+\alpha/K$, $\alpha \in (0,1)$

[4]ResNet and Inception are two benchmark deep learning models in computer vision context. Since they are not related to this thesis, the introduction to them is skipped. CIFAR-100 and ImageNet are two benchmark datasets in computer vision context.

is a hyperparameter. LS replaces the one-hot label (so called hard label) y_i with the mixture of y_i and a uniform distribution α/K (y_i^{LS} is also called soft label). If $\alpha \to 0$, $y_i^{LS} \to y_i$. If $\alpha \to 1$, the distribution of y_i^{LS} approximates the uniform distribution.
LS could prevent the model from becoming overconfident by artificially designing soft targets. This also means the calibration of the model could also inherit this benefit since LS makes the predictions more accurately represent the accuracy. Temperature scaling (as shown in Eq. 2.36) has ready been shown to help improve calibrate the model [16]. Muller et al. showed that LS has a similar effect with temperature scaling. Fig. 2.3 shows that with $\alpha = 0.05$ (left plot) and $\alpha = 0.1$ (right plot), models are similarly calibrated compared to temperature scaling $T = 1.9$ (left plot) and $T = 1.4$ (right plot). Both of them improved calibration performance independently compared to the vanilla model (making the line closer to the diagonal line in reliability diagram).

Chapter 3

Analysis

In this chapter, we first investigate the calibration performance on benchmark datasets with benchmark GNN models, and how existing state-of-the-art calibration methods help calibrate. Then we analyze the influence of GNNs model properties and dataset characteristics.

3.1 Analysis of calibration

3.1.1 Datasets

Cora, Citeseer and Pubmed [49] are the most widely-used benchmark classification datasets in recent published GNNs papers. They are three citation networks, where nodes represent papers and the edges represent a citation between papers (undirected edges). Node features are textual features, or bag-of-words. Each node has a d-dimensional feature vector. The predicted label of nodes is the topic of that paper. The statistics of the dataset can be viewed in Table 3.1.

Table 3.1: Dataset statistics for citation datasets. Label rate denotes the training node number divided by the total node number.

Dataset	K	d	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	Label rate
Citeseer	6	3703	3327	4732	0.036
Cora	7	1433	2708	5429	0.052
Pubmed	3	500	19717	44338	0.003

3.1.2 GNN models, metrics and methods

In our experiments, we employed GCN [21], GAT [25], SGC [10], gfNN [22], APPNP [50].
Calibration metrics used include ECE and marginal-ECE. We choose these two metrics since ECE is quite widely-used and convenient to compare among different methods, and marginal-ECE is a suitable metric for all classes' calibration.
Models, metrics and methods are introduced in Chapter 2.

3.1.3 Experiment setup and hyperparameters

For citation datasets, hyperparameters of GCN, GAT, SGC and APPNP are taken from their best values in Pytorch geometric library[1] [51]. For SGC, we set $lr(learningrate) = 0.2$, training for 100 epochs and tune weight decay for 60 iterations using hyperopt[2] [52], like the original paper. We train SGC without early-stopping. For gfNN, we take $lr = 0.2$, $hid.dimension = 32$, $epochs = 100$, and $layernum. = 2$. We train with early-stopping patience of 10. Weight decay is tuned for 60 iterations using hyperopt. Each model is run in averaged 100 times.

3.1.4 Results and discussions

3.1.4.1. Uncalibrated performance

In Table 3.2, we show the uncalibrated performance with respect to accuracy, ECE and marginal-ECE.
GNNs exhibit underconfidence. Reliability diagrams and confidence histograms of GCN on Citeseer are shown in the left plots in Fig. 3.1. The figures of the other models and datasets are shown in Appendix A. In uncalibrated reliability diagrams, the avg. confidence in most bins is smaller than accuracy.
The method APPNP has the best performance in terms of accuracy. APPNP, GAT and gfNN have the best performance in terms of calibration. More-

[1] https://github.com/pyg-team/pytorch_geometric
[2] https://github.com/hyperopt/hyperopt

Table 3.2: Uncalibrated performance of all GNN models with respect to accuracy, ECE, and marginal ECE (mean±standard deviation over 100-time runs). The best result for each dataset and metric is displayed in bold.

Dataset	Model	Acc.	ECE	Marg. ECE
Cora	GCN	81.34±0.47	21.17±0.73	6.50±0.16
	GAT	83.26±0.41	**15.19±0.54**	**5.08±0.20**
	SGC	80.98±0.05	24.95±0.19	7.62±0.06
	gfNN	80.49±0.90	16.30±0.35	5.24±0.08
	APPNP	**83.44±0.44**	16.96±0.56	5.61±0.18
Citeseer	GCN	71.24±0.65	21.04±0.89	8.49±0.25
	GAT	70.90±0.56	17.00±0.55	7.61±0.27
	SGC	71.91±0.05	42.89±0.08	15.54±0.01
	gfNN	70.50±1.32	20.06±0.54	8.15±0.13
	APPNP	**72.04±0.46**	**13.24±0.57**	**6.24±0.32**
Pubmed	GCN	79.17±0.46	6.59±0.65	5.30±0.51
	GAT	78.22±0.41	4.46±0.76	4.70±0.75
	SGC	78.32±0.16	20.99±0.02	13.84±0.01
	gfNN	79.24±0.45	**4.38±0.42**	**3.79±0.24**
	APPNP	**80.13±0.29**	5.14±0.83	5.27±0.92

over, we find that GAT always outperforms GCN in terms of ECE and MECE.

3.1.4.2. Calibration performance.

Results of calibration methods are shown in Table 3.3. We can see that all post-processing methods could effectively calibrate GNN models. Among them, histogram binning and isotonic regression perform the best.

For a calibration method, besides a good calibration performance, accuracy-preservation or accuracy-improvement are also desirable properties, therefore we present results of calibrated accuracy for each method, as shown in Table 3.4. Temperature scaling does not change the order of scores, therefore does not change the accuracy. Other methods can reach an equal or even improved accuracy.

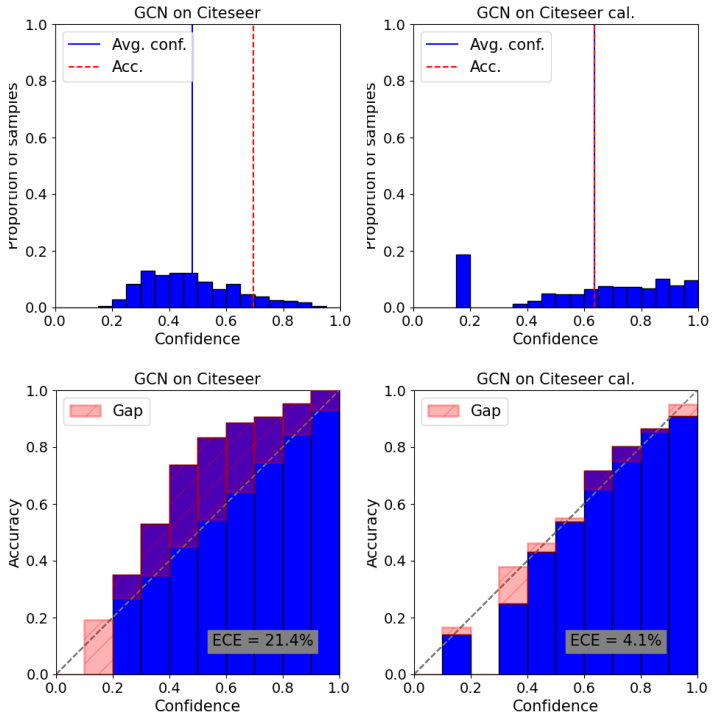


Figure 3.1: Reliability diagrams (second row) and confidence histograms (first row) for GCN on Citeseer. The left two plots are uncalibrated and the right two are calibrated using histogram binning. The diagonal line indicates perfect calibration.

3.2 Analysis of model architecture components

3.2.1 Model depth

In this experiment, we investigate the influence of model depth, i.e., the number of layers, on calibration performance.

Setting. We run experiments on the influence of depth on the models GCN and GAT. Experiment setting follows by Appendix B in Kipf et al. [21]. We set the number of layers in GCN and GAT from 1 to 10. In GAT, the number of attention heads is set to 1 for comparison. The number of units for each hidden layer is 16. We train for 200 epochs with early stopping of patience of 10. Dropout layers are removed. The other parameters are kept the same as Section 3.1.3. Results are over 10 independent runs.

Results and discussions. As Fig. 3.2 and and Appendix A shown, first, over-smoothing (introduced in Chapter 2) becomes obvious when models

Table 3.3: Calibrated performance with respect to ECE (mean±standard deviation over 100 independent runs). The best GNN model for each calibration method is underlined. The best calibration method for each GNN model is displayed in bold.

Dataset	Model	Uncal.	His. bin.	Iso. reg.	BBQ	Tem. scal.	Meta-Cal
Cora	GCN	21.17±0.73	4.42±0.77	3.92±0.71	4.50±0.73	**3.73±0.62**	3.88±0.60
	GAT	15.19±0.54	4.83±0.56	4.07±0.57	4.12±0.55	3.75±0.73	**3.37±0.65**
	SGC	24.95±0.19	4.89±0.30	**3.63±0.27**	4.25±0.43	3.95±0.18	4.02±0.40
	gfNN	16.30±0.35	3.18±0.59	**3.08±0.43**	3.74±0.45	3.22±0.49	3.53±0.45
	APPNP	16.96±0.56	4.37±0.67	3.78±0.59	3.99±0.69	**3.38±0.59**	3.67±0.66
Citeseer	GCN	21.04±0.89	**4.58±0.76**	4.62±0.92	5.44±1.00	4.87±0.67	4.93±0.74
	GAT	17.00±0.55	5.32±0.75	**5.19±0.68**	5.27±0.73	5.88±0.64	6.27±0.65
	SGC	42.89±0.08	4.63±0.21	**3.94±0.15**	5.94±0.31	4.47±0.15	4.50±0.23
	gfNN	20.06±0.54	**4.40±0.66**	4.52±0.75	4.55±0.70	5.11±0.49	4.76±0.56
	APPNP	13.24±0.57	4.96±0.92	**4.85±0.84**	5.47±0.87	5.00±0.78	5.40±0.73
Pubmed	GCN	6.59±0.65	5.06±0.80	4.85±0.74	4.98±0.83	**4.38±0.60**	5.03±0.79
	GAT	4.46±0.76	4.77±0.86	4.73±0.77	4.97±0.86	**3.96±0.69**	4.56±1.09
	SGC	20.99±0.02	4.43±0.20	**3.74±0.29**	4.00±0.21	4.46±0.19	4.41±0.69
	gfNN	4.38±0.42	5.35±0.82	4.98±0.65	**4.74±0.67**	4.83±0.52	6.18±0.82
	APPNP	5.14±0.83	4.90±0.77	4.47±0.60	4.74±0.78	**4.34±0.75**	5.11±0.95

get deeper than 3 layers and accuracy starts to decrease. The best results for both classification and calibration are obtained with 2-3 layers. ECE first increases due to the smoothing effect, then decreases since the accuracy is decreased significantly.

3.2.2 Model width

Model depth might cause oversmoothing for GNNs, while model width is a factor of model capacity. We compare the influence of model width, i.e., the number of neurons per layer, to calibration.

Setting. We run experiments on the influence of width on the models GCN and GAT. The width varies in $\{8, 16, 32, ..., 1024\}$. The other parameters are kept the same as Section 3.1.3. Results are over 10 independent runs.

Results and discussions. Results are shown in Fig. 3.3 and Appendix A. Wider nets almost keep the accuracy as a constant but could help calibrate.

Table 3.4: Calibrated accuracy (mean±standard deviation over 100 independent runs) for each calibration method and GNN model. Temperature scaling and RBS do not change the accuracy.

Dataset	Model	Uncal.	His	Iso	BBQ	Meta
Cora	GCN	81.34±0.47	80.22±0.70	81.55±0.53	81.17±0.71	78.46±1.87
	GAT	83.26±0.41	81.43±0.46	83.88±0.41	83.43±0.51	79.13±1.56
	SGC	80.98±0.05	79.65±0.09	81.57±0.18	80.92±0.07	78.70±1.73
	gfNN	80.49±0.90	79.61±0.34	80.63±0.28	80.68±0.51	78.12±1.93
	APPNP	83.44±0.44	82.75±0.58	83.70±0.54	83.19±0.67	80.73±1.67
Citeseer	GCN	71.24±0.65	72.01±0.77	72.50±0.75	71.98±0.99	68.12±1.95
	GAT	70.90±0.56	71.59±0.61	71.97±0.53	71.51±0.70	67.17±2.02
	SGC	71.91±0.05	73.26±0.14	74.04±0.08	72.49±0.15	69.42±1.76
	gfNN	70.50±1.32	72.9±0.59	72.72±0.53	71.36±0.65	69.15±1.76
	APPNP	72.04±0.46	72.95±0.51	72.68±0.48	72.54±0.79	68.98±1.92
Pubmed	GCN	79.17±0.46	78.42±0.71	78.81±0.54	78.64±0.85	77.05±1.70
	GAT	78.22±0.41	77.09±0.45	77.62±0.39	76.84±0.44	75.27±2.06
	SGC	78.32±0.16	78.99±0.05	78.98±0.12	79.09±0.12	77.86±1.64
	gfNN	79.24±0.45	77.99±0.49	78.77±0.33	78.47±0.39	76.74±1.66
	APPNP	80.13±0.29	79.93±0.45	80.15±0.37	79.80±0.62	78.09±1.88

3.2.3 Aggregation

In this experiment, we investigate the influence of aggregation of neighbors on calibration performance. In message-passing GNNs, node i's aggregation is represented in the edge of node i to its neighbors. If we remove the edge of node i, its aggregation is decreasing. To this end, we remove the proportion of edge randomly from 0% (as normal) to 100% (no graph structure), and observe the accuracy and ECE performance of the models GCN and GAT.
Setting. Metrics we use here include accuracy, ECE, marginal-ECE and classwise-ECE[3]. The other parameters are kept the same as Section 3.1.3. Results are over 10 independent runs.
Results and discussions. Results are shown in Fig. 3.4. With removed aggregation (avg. num. of edges per node), GCN and GAT gain worse accuracy then without removing aggregation. In terms of ECE, marginal-ECE and classwise-ECE, GCN is relatively stable but GAT's neighborhood

[3]Classwise-ECE is a multi-class calibration error metric from Kull et al. [53]. It's very similar to marginal-ECE (see Chapter 2). The only difference is that at implementation, marginal-ECE takes equal-number bins while classwise-ECE takes equal-space bins.

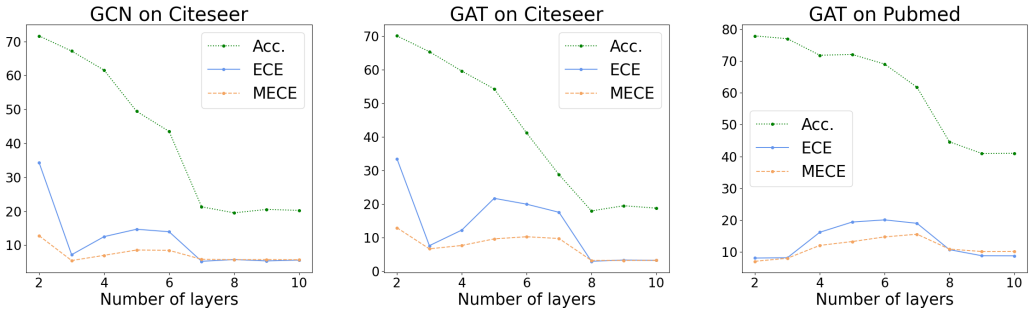


Figure 3.2: Varying depth (number of layers) for GCN and GAT.

aggregation can help calibrate. Attention of edges to neighbors (introduced in Chapter 2) brings in helpful information to calibration.

3.2.4 Regularization

Regularization methods used to mitigate overfitting in deep learning also intuitively relate to calibration [16]. Some regularization methods are already shown to help calibration [48, 8, 45]. But the exact relationship and the best approach to prevent overfitting with respect to calibration remain an open question.

Dropout is a simple but powerful regularization method proposed by Srivastava et al. [54]. It is implemented by randomly ignoring some number of layers or some neurons in a layer during training with a fixed rate. Then at testing, this dropout is removed. All layers and neurons participate in the model. We compare the influence of regularization, i.e., dropout, to calibration.

Setting. We run experiments on the influence of regularization on the models GCN and GAT, with 4 conditions of dropout without dropout, with dropout before the layer, after each layer, and on all places, keeping fixed dropout rate as 0.5. The other parameters are kept the same as Section 3.1.3. Results are over 10 independent runs.

Results. Adding dropout to the model could improve accuracy at times but hurt ECE in almost all cases, shown in Fig. 3.5. This hurting calibration conclusion is universal with models and datasets. This is consistent with one conclusion from Guo et al.[16], "Modern neural networks exhibit

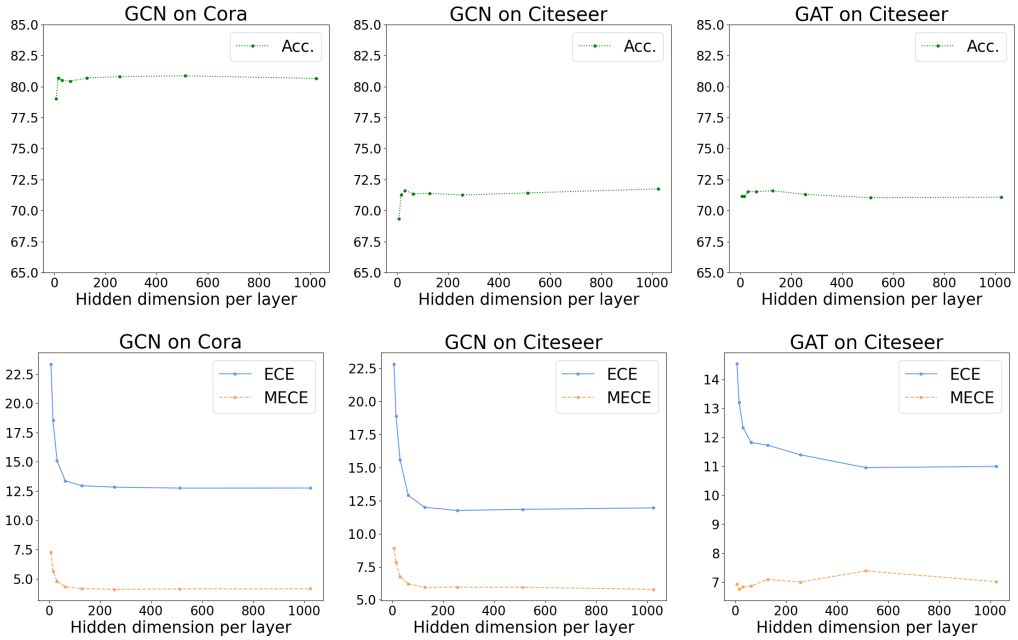


Figure 3.3: Varying model width (hidden dimensions per layer).

a strange phenomenon: probabilistic error and miscalibration worsen even as classification error is reduced.”.

3.3 Analysis of dataset characteristics

In this part, we explore the influence of dataset characteristics, i.e. the classes are imbalanced or balanced, to calibration. Balancing classes is extremely important for imbalanced tasks where the model cannot learn the meaningful information. A common balancing method is to simply weight the loss function, such that each example contributes the same to the loss. And it is also important to balance the test data (since otherwise predictions and patterns may change immensely from training to testing [55].), and balance the metrics.

In the context of GNNs, Teixeira et al. shows that by balancing the classes for a difficult and severe class imbalance task (Friendster[4]), it could lead

[4]Friendster is a social dataset proposed by Teixeira et al. [18]. Simialr to citation datasets introduced before, the node represents users and the edge represents friendship. The features include users’ ages, gender, interests, etc, encoded as one-hot features. The label of a node is the relationship of the user:

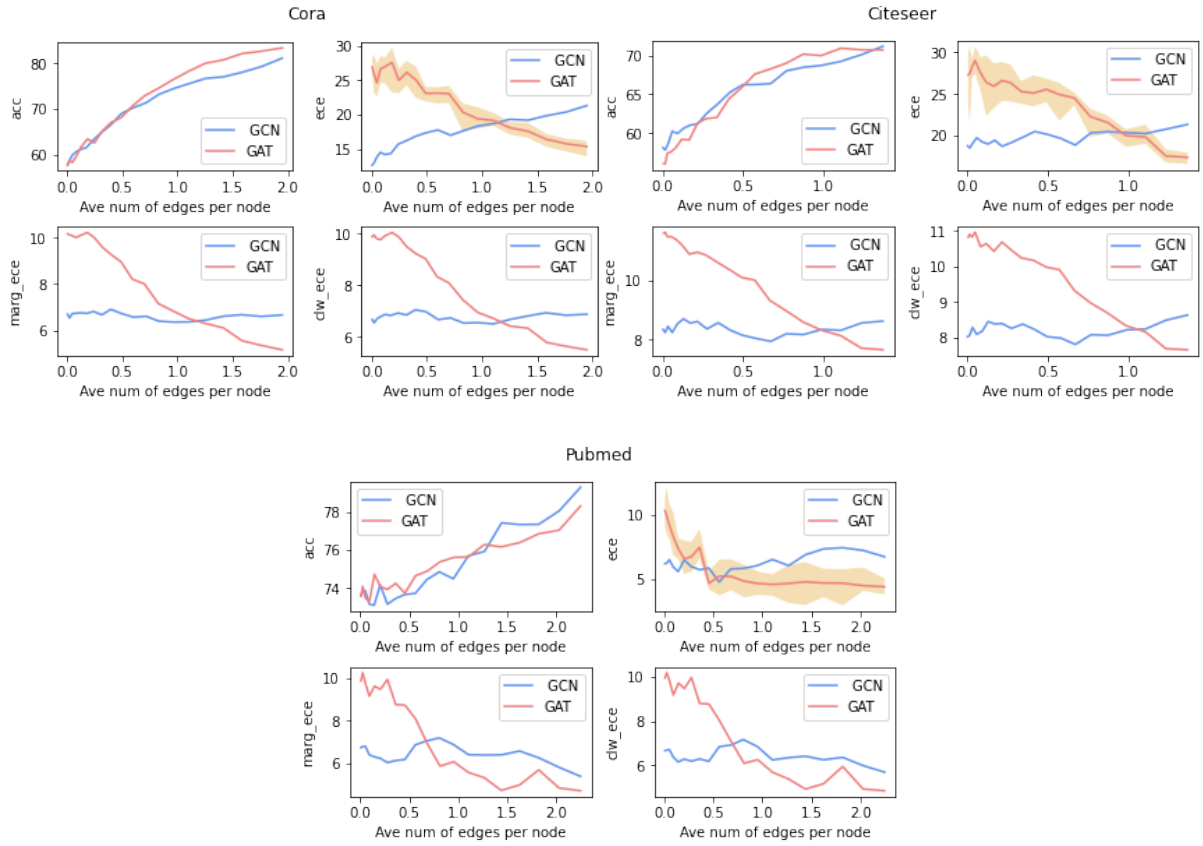


Figure 3.4: Aggregation's influence in GCN and GAT. Here we only show uncertainty shading for ECE of GAT.

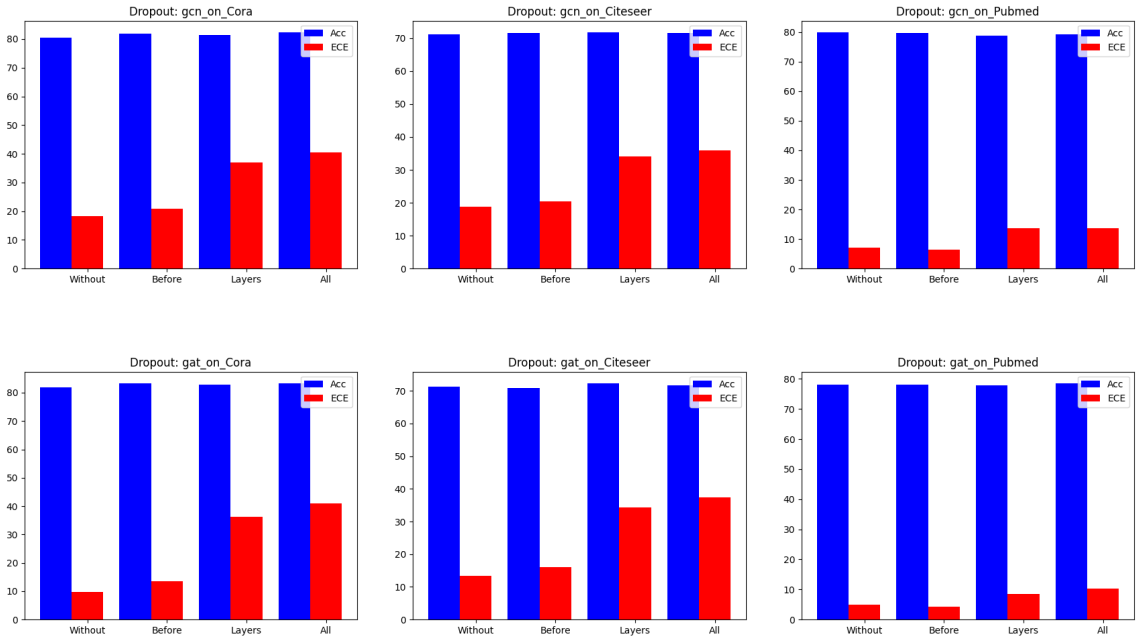


Figure 3.5: Dropout in different places for GCN and GAT. From left to right in each row: without dropout, with dropout before the layer, after each layer and on all places.

to a more calibrated result [18] . They further show that such a balancing method could lead to a better calibrated model than using calibration methods like temperature scaling and etc.

But that observation only appears in their hardest dataset (Friendster). Our experiments here explore the influence of balancing classes in benchmark citation datasets.

Recall that in Section 3.1.4, we show the imbalance result for citation datasets (see Table. 3.2). To balance classes and have a fair comparison, besides balancing training and test data (by balancing loss functions), we also need to balance metrics. Such an balancing methods (train and test loss, test accuracy and calibration metrics) could have an equivalent effect as balancing the whole dataset, therefore it could be a fair comparison before and after balancing.

As for balancing metrics, here we define a new metric called *Balanced-ECE*. Intuitively, Balanced-ECE is an balancing-classes extension of ECE: it is a

single, married, in relationship and in domestic relationship.

Table 3.5: Dataset imbalance statistics for citation datasets. Here imbalance ratio denotes the number of largest classes/the number of the smallest classes.

Dataset	K	Imbalance ratio
Citeseer	6	2.82
Cora	7	4.54
Pubmed	3	1.92

top-label calibration metric, although it looks similar to marginal-ECE (see Chapter 2). Balanced-ECE is defined as:

$$Balanced\text{-}ECE = \sum_{m=1}^{M}\sum_{k=1}^{K} w_k \cdot \frac{|B_m|}{N}|acc(B_{mk}) - conf(B_{mk})|, \tag{3.1}$$

where M is the number of bins, N being the number of samples, $|B_m|$ being the number of samples in m-th bin, B_{mk} being the number of k-th class samples in m-th bin, w_k being a weighted matrix for the class k, denoting how important the class k is. If all classes are equally important, $w_k = 1/k$. $acc(B_{mk})$ and $conf(B_{mk})$ are accuracy of samples in m-th bin with k-th class true label, and averaged confidence scores of those samples respectively.

Setting. We run experiments on the influence of balancing classes on the GNN model family. Data distribution is balanced in both training and test losses. Both accuracy and calibration metrics are balanced. The other parameters are kept the same as Section 3.1.3. Results are over 10 independent runs.

Results and discussions. Results can be seen in Table 3.6. Compared with Table 3.2, it shows that for not very imbalanced citation datasets, balancing classes cannot provide obvious improvement to calibration performance, neither accuracy. For such datasets, GNNs have the ability to automatically balance classes

But a claim can also be made that our citation datasets are not very imbalanced (since the imbalanced ratio is not very high in Table 3.5), future work could be done when considering more imbalanced datasets (e.g. Amazon-computer [28] with imbalanced ratio of 18 and Cora-Full [28] with imbalanced ratio of 62).

Table 3.6: (Balanced uncalibrated performance) Accuracy, uncalibrated ECE, Marginal ECE, class-wise ECE (w/ std. deviation over 10-time independent runs) for each GNN family model. The best result for each dataset and metric is displayed in bold.

Datasets	Model	Acc.	ECE	Marg. ECE
Cora	GCN	81.29±0.59	21.76±0.77	6.52±0.18
Cora	GAT	83.17±0.34	**16.24±0.67**	**5.07±0.2**
Cora	SGC	80.75±0.07	39.85±0.15	11.8±0.03
Cora	gfNN	80.01±0.95	19.73±1.54	6.22±0.62
Cora	APPNP	**83.66±0.54**	17.97±0.71	5.61±0.18
CiteSeer	GCN	68.52±0.78	22.3±0.96	8.49±0.25
CiteSeer	GAT	67.83±0.48	20.49±0.89	7.61±0.26
CiteSeer	SGC	68.48±0.01	41.52±0.02	16.09±0.02
CiteSeer	gfNN	67.5±1.01	18.97±2.23	7.97±1.33
CiteSeer	APPNP	**68.85±0.53**	**17.59±0.69**	**6.24±0.33**
Pubmed	GCN	79.15±0.47	7.73±0.66	5.3±0.52
Pubmed	GAT	78.4±0.41	**7.66±0.74**	**4.69±0.74**
Pubmed	SGC	78.05±0.17	21.14±0.07	14.13±0.02
Pubmed	gfNN	78.75±0.8	10.42±8.06	6.2±5.57
Pubmed	APPNP	**80.25±0.38**	8.01±0.68	5.27±0.91

3.4 Conclusion

We provide a comprehensive analysis of GNNs and node classification datasets in this chapter.

1. We first show that benchmark GNNs on citation datasets exhibit underconfidence. Existing calibration methods lead to well-calibrated results.

2. Further, we show that model depth, width, aggregation and regularization could all affect models' calibration.

2.1. When increasing depth, GNNs becomes first less calibrated, then more calibrated due to a lower accuracy.

2.2. Wider networks can help calibrate.

2.3. Neighborhood aggregation with attention could help calibrate.

2.4. Miscalibration worsens when regularization is used to reduce classification errors.

3. GNNs perform stably in terms of accuracy and calibration as dataset imbalance is not very severe.

Chapter 4

Knowledge distillation for calibration

In this chapter, first we give an introduction about the basic knowledge distillation, and one knowledge distillation method on graph representation learning. Then we investigate whether knowledge distillation could help calibrate.

4.1 Knowledge distillation

4.1.1 The basic knowledge distillation

Intuition. Knowledge Distillation (KD) [47] was first proposed by Hinton et al.. Basically, a KD system consists of three key components: knowledge, distillation algorithm, and teacher-student architecture, which is shown in Fig. 4.1. The goal is to distill the knowledge from a usually large teacher model into a smaller model so that the student model can hold a similar performance to the big one.
Knowledge distillation is inspired by the intuition that the relative prob-

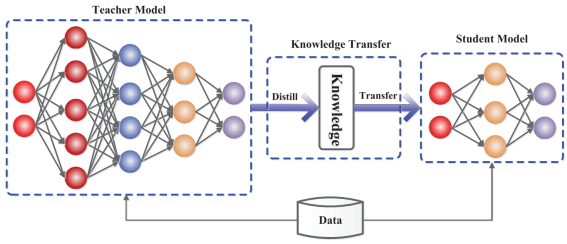


Figure 4.1: An example schematic for knowdge distillation. [9]

abilities of incorrect answers also tell us a lot [47]. Usually, the training objective of a benchmark model is to maximize the average log probability of the correct answer, but a side-effect is that the trained model assigns probabilities to all the incorrect answers to make them smaller to make them less different from each other. However, these smaller probabilities also contribute to the final performance, since some of these small values are still higher than the others. For example, a Shiba Inu image may have a small wrong probability of being a dog of another category, which is still many times more probable than being a car.

KD loss. A way to transfer the generalization ability from a big model to a small one is to use the probabilities of the big model as soft targets for training the small model. The teacher's output logit vector z is smoothed by setting a temperature T to the gain the smoothed logits $q(z)$. This is the same form as temperature scaling as shown in Eq. 2.36:

$$q(z) = \sigma(\frac{z}{T}) = \frac{exp(\frac{z}{T})}{\sum_{i=1}^{K} exp(\frac{z_i}{T})}, \tag{4.1}$$

where T is usually higher than 1 to soften z.

Given two probability predictions p and q, let p_i and q_i denote their probability for sample i, Kullback–Leibler divergence (KL divergence) is defined as:

$$L_{KL}(p|q) = \sum_{i}^{N} p_i log \frac{p_i}{q_i} \tag{4.2}$$

The more similar p and q are, the smaller $L_{KL}(p|q)$ is.

Let y_i and $\hat{p}_i$ denote true label and predicted confidence for the distilled model for sample i.

For the distilled model, the loss function is a weighted average of two different loss terms. The first loss term is the KL divergence of the distilled model's prediction p and the soft prediction q from the cumbersome model, $L_{KL}(p|q)$. The second term is the cross-entropy loss for the distilled model, $L_{CE}(p, y) = -\sum_{i=1}^{N} y_i \log \hat{p}_i$. At the same time, according to Ref. [47], it is important to multiply the first loss with T^2, since the magnitude of the gradients produced by the scale of the soft targets scale as $1/T^2$. In this way, it ensures that when the temperature changes, the relative contribu-

tions of these two losses remain roughly unchanged. Above all, if we write more precisely, the total loss can be written as:

$$L = \lambda \cdot T^2 \cdot L_{KL}(p|q) + (1 - \lambda)L_{CE}(p, y). \tag{4.3}$$

4.2 KD for graph representations

Most KD methods focus on Convolutional Neural Networks (CNNs) in the context of computer vision[9]. In comparison, less attention was devoted to combining KD with GNNs.

4.2.1 Combination of Parameterized label propagation and Feature transformation (CPF)

CPF [56] is a KD method designed on graphs and combining GNNs. It is motivated by the issue that GNNs could not make full use of structure knowledge lying in the data [57]. Treating all messages from neighbors as a set is one kind of this issue. For example, GCN simply aggregates information from all one-hop neighbors to the central node in an iteration [21]. Such a process cannot distinguish the message from different nodes, and therefore loses the structural information. The solution to address it is to design a KD framework. In this framework, the student model includes two modules: label propagation module [58] and feature transformation module. Then a final loss function is to minimize the difference between pre-trained teacher GNN predictions and student GNN predictions.

Label propagation. *label propagation* (LP) is a module used here to preserve the structure-based prior knowledge. LP is a model designed to learn labels for unknown nodes. It is based on the *homophilic graph assumption*: nodes linked by edges tend to have same labels. LP uses this assumption to propagate node labels iteratively from labeled nodes to unlabeled nodes. Formally, we denote the labeled node set as V_L and unlabeled node set as V_U. V_L and $V_U \in \mathcal{V}$. For node v, if we know its label, we set its LP prediction as the ground-truth one-hot label. Otherwise, we set it as a uniform distribution among K classes. We set the node v's initial prediction

of LP, $f_{LP}^0(v)$ as:

$$f_{LP}^0(v) = \begin{cases} (0, ..., 1, ...0) \in \mathbb{R}^K, & for\ v \in V_L \\ (\frac{1}{K}, ...\frac{1}{K}, ...\frac{1}{K}) \in \mathbb{R}^K, & for\ v \in V_U \end{cases} \tag{4.4}$$

We denote node v's LP prediction after m iterations as f_{LP}^m. The update function for $m+1$ layer can be written as:

$$f_{LP}^{m+1}(v) = \sum_{u \in N_v \cup \{v\}} w_{uv} f_{LP}^m(u), \tag{4.5}$$

where w_{uv} is the edge weight between node u and node v computed by:

$$w_{uv} = \frac{exp(z^T X_u)}{\sum_{u' \in N_v \cup \{v'\}} exp(z^T X_{u'})}, \tag{4.6}$$

where $z \in \mathbb{R}^d$ is a learnable parameter, $X_u \in \mathbb{R}^d$ being the feature vector for node u. $z^T X_u$ is called confidence score for node u.

Feature transformation. The above LP module propagates label according to the graph structure. Besides LP module, there is also a designed feature transformation module (FT) to emphasize the node features. To this end, a simple model is designed with an input node feature vector, outputting encoded feature information for this node. Node v's FT is simply a two-layer MLP of node v's feature vector X_v then applying a softmax function:

$$f_{FT}(v) = softmax(MLP(X_v)). \tag{4.7}$$

The final score and KD loss. Combining LP module and FT module, a final score for node v can be learnt. To this end, a trainable parameter $\alpha_v \in [0, 1]$ is designed to balance the component of predictions from both modules. More formally, after $m+1$ iterations, the final CPF score $f_{CPF}^{m+1}(v)$ is:

$$f_{CPF}^{m+1}(v) = \alpha_v \sum_{u \in N_v \cup \{v\}} w_{uv} f_{CPF}^m(u) + (1 - \alpha_v) f_{FT}(v), \tag{4.8}$$

where the initial CPF score is set as the M-th iteration LP score: $f_{CPF}^0(v) = f_{LP}^M(v)$, and edge weights w_{uv} is chosen the same as w_{uv} in LP module in Eq. 4.5.

The CPF KD loss function is to minimize the difference between teacher's prediction $f_{GNN}(v)$ and student's CPF prediction after M iterations $f^M_{CPF}(v)$,

$$L_{CPF} = \sum_{v \in V_U} ||f_{GNN}(v) - f^M_{CPF}(v)||_2, \tag{4.9}$$

where $||...||_2$ denotes L2-norm.

4.3 KD for calibration

Although a lot of KD work have been proposed, none of them discussed the calibration performance of KDs. Our work here first aims to evaluate the influence of KD to calibration on GNNs.
The first intuition to use KD for calibration is that by learning from the teacher's "softer" logits, students could learn more information about the model's decision-making and confidence estimation, which should be useful to calibration.
The second intuition is that, Muller et al. [8] discovered that label smoothing (introduced in Chapter 2) improves calibration implicitly. At the same time, Li et al. [19] proposed to view KD as an adaptive version of label smoothing, and therefore KD should inherit the benefit of label smoothing. Besides, they interpreted KD as a regularization term, which also should help prevent model probability overfitting and help calibrate.

4.3.1 Empirical studies of calibration performance of KDs on graphs.

4.3.1.1 A toy experiment.

A first try to evaluate the influence of hyperparameters KD to calibration could be done easily. We distill the information from a 4-layer GCN teacher model to a student that has the same architecture as the teacher. In the formula of KD, $L = \alpha \cdot T^2 \cdot L_{KL}(p|q) + (1-\alpha)L_{CE}(p,y)$, we evaluate the influence of hyperparameters of α and T as KD, with α chosen in $\{0.99, 0.95, 0.5, 0.1, 0.05\}$ and T chosen in $\{1, 2, 5, 10\}$, with the same setting as in [59].

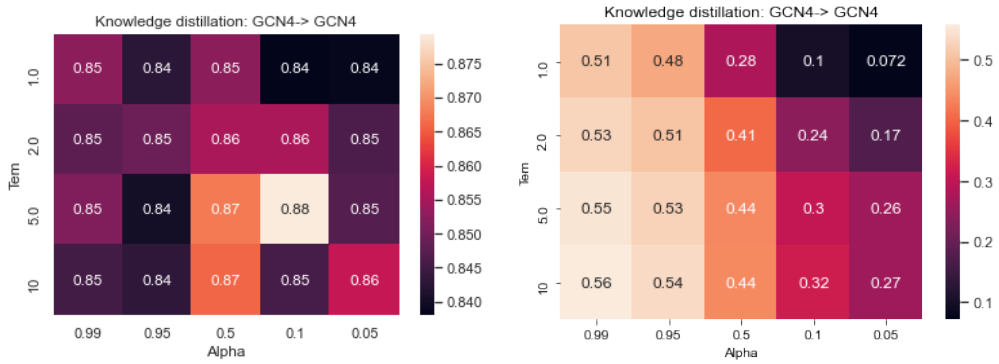


Figure 4.2: Evaluating KD hyperparameters on Cora. Here the teacher has accuracy of 0.84 and ECE of 0.05. Left: Accuracy. Right: ECE. Note the train-val-test split is not the same as experiments before.

Results. Results can be viewed in Fig. 4.2, where the left plot is the influence to student accuracy, the right plot is to ECE. Since it is only a one-time running experiment, the result might have high variance, but still clearly showed by choosing optimal α and T, the student can obtain a higher accuracy than the teacher[1]. Like the original paper suggested, a small α generally can lead to better performance [47]. While for ECE, the student consistently performs worse than the teacher with different parameter combinations.

4.3.1.2 Basic KD and CPF experiments.

More KD benchmark methods and datasets were utilized to explore whether KD can help calibrate. The baseline methods we choose include the basic KD and CPF.

Setting. For KD methods, hyperparameter tuning could be an important factor in the final result. Here we state our choice of parameters in each method. For the basic KD, we take $\alpha = 0.1$, the same as the original paper. T is tuned in validation set in $\{1, 5, 10, 50, 100\}$. We use Adam optimizer like the original GNNs papers with $lr = 0.01$ and weight decay tuned in the range of $[1e-9, 1e-3]$. Teacher and student models are repeated 10 times and averaged. For CPF, we trained the teacher model one time and trained student models with 10 time-averaged learning from the same

[1]In GNNs-related experiments, even a 1% accuracy improvement is considered as an achievement.

Table 4.1: CPF of GCN and GAT on Cora, Citeseer, Pubmed, Amazon-computers and photos datasets. Acc./ECE ↑ denotes the acc./ECE increasing rate for students.

Datasets	Teacher	Teacher Acc.	Student acc.	Acc. ↑ (%)	Teacher ECE	Student ECE	ECE ↑ (%)
Cora	GCN	82.6	83.63±0.48	1.2	25.4	38.27±1.79	50.7
Citeseer	GCN	69.2	72.33±0.56	4.5	6.63	33.09±7.60	399.1
Pubmed	GCN	79.8	75.93±7.58	-4.8	9.02	27.33±1.95	203.0
A-computers	GCN	83.95	84.16±0.87	0.3	4.98	5.62±0.65	12.9
A-photos	GCN	90.88	93.26±0.34	2.6	6.46	14.86±2.91	130.0
Cora	GAT	81.7	83.18±0.44	1.8	17.77	31.80±2.12	79.0
Citeseer	GAT	70.7	72.68±0.75	2.8	35.13	44.30±2.88	26.1
Pubmed	GAT	76.6	72.21±6.60	-5.7	32.46	31.24±3.50	-3.8
A-computers	GAT	79.86	80.66±0.39	1.0	25.85	27.97±0.61	8.2
A-photos	GAT	89.53	90.59±0.60	1.2	21.53	21.96±1.14	2.0

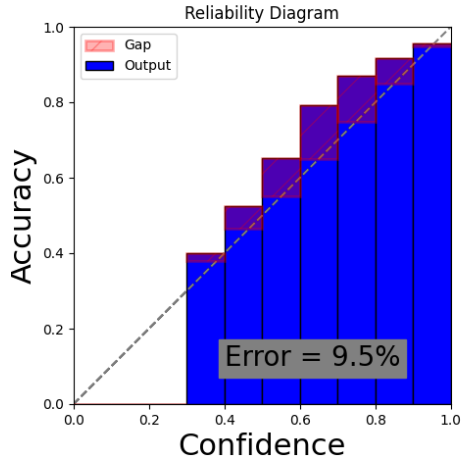
teacher. Training epochs of the teacher and students are 500, with early-stopping of 50 patience. Hyperparameters are taken from the original paper [56].

Datasets. We utilize three commonly used benchmark datasets, Cora, CIteseer and Pubmed [49] as in Chapter 3. For CPF, we also utilize two more datasets, Amazon-computer and Amazon-photos [28]. They are items co-puchased graph in Amazon, like mentioned in Section 2.1.3.5. These two datasets' nodes represent goods and edges represent they are frequently co-purchased. Features are information about the goods and labels denote their categories. The task is to predict unseen nodes' labels.

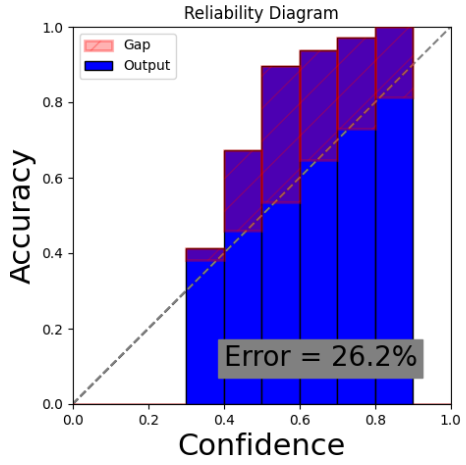
Results. Results are shown in Table. 4.1 (CPF) and Table. 4.2 (the basic KD). Both basic KD and CPF could help improve accuracy for most cases but at the cost of hurting ECE. The accuracy decrease of CPF on Pubmed is due to the hyperparameter is not stable enough. For other cases, CPF can obtain a clear increase of accuracy and ECE. For instance, CPF on Cora has an $1\% - 2\%$ accuracy improvement for GCN and GAT models, but at the cost of $50\% - 80\%$ ECE increasing. If we further observe the confidence histograms, we will see that the knowledge distillation leads to a more underconfident student model, as shown in Fig. **??**. A further discussion is as follows.

Table 4.2: KD of GCN and GAT on Cora, Citeseer, and Pubmed datasets.

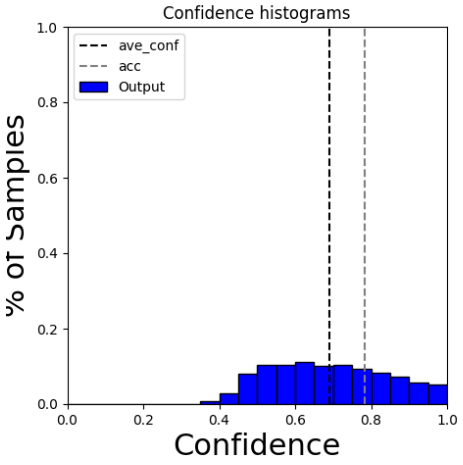
Datasets	Teacher	Teacher acc	KD	Acc ↑ (%)	Teacher ECE	KD	ECE ↑ (%)
Cora	GCN	81.07±0.31	81.45±0.26	0.5	21.32±0.94	19.64±0.94	-7.9
Citeseer	GCN	71.17±0.6	71.33±0.45	0.2	21.32±1.16	21.11±1.03	-1.0
Pubmed	GCN	79.32±0.32	79.07±0.37	-0.3	6.73±0.71	8.38±0.75	24.5
Cora	GAT	82.99±0.45	83.2±0.39	0.3	15.11±0.72	19.27±0.49	27.5
Citeseer	GAT	70.7±0.28	71.55±0.38	1.2	17.36±0.39	26.22±0.7	51.0
Pubmed	GAT	78.2±0.39	78.14±0.15	-0.1	4.42±0.67	4.34±0.58	-1.8



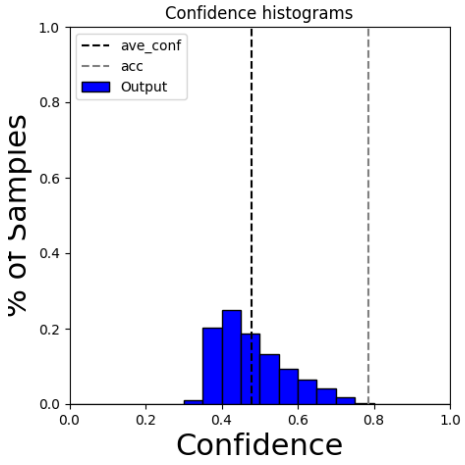
(a) CPF teacher, GCN on Pubmed



(b) CPF student, GCN on Pubmed



(c) CPF teacher, GCN on Pubmed



(d) CPF student, GCN on Pubmed

Figure 4.3: Confidence histograms and reliability diagrams for CPF. Take an example of GCN on Pubmed. Left: teacher. Right: student.

4.3.2 A close look at KD and student calibration performance

Yuan et al. [19] explores the relationship between KD and label smoothing (LS) [8] and found that LS (LS was introduced in Section 2.3.2.) can be interpreted as ad-hoc KD, with teacher's probability distribution is uniform and $T = 1$. Therefore, KD should inherit the benefits of label smoothing, i.e. helping prevent models from being overconfident, for modern overconfident neural networks. But if the teacher model is underconfident, KD, or label smoothing, leads to a more underconfident result, i.e. our GNNs case.

Claim 1. *Suppose the student model has the same architecture as the teacher, KD helps calibrate an original overconfident model, and miscalibrate an original underconfident model.*

Claim 1 could be understood by that students trained by LS-produced soft labels (recall the LS-produced soft label $y_i^{LS} = y_i(1-\alpha) + \alpha/K$, where $\alpha \in (0, 1)$) has an equivalent effect of *assigning a smoothing score α/K to incorrect classes' labels* (which should be 0 in one-hot labels), and therefore push down the top-class score. *KD's effects should be similar to and smaller than LS's since KD produces "not that soft" labels.*

Note that for models trained using cross-entropy, trained largest probability inherently is pushed to approach the model label (If it is with hard label, the largest probability is pushed to approach to 1 and *usually* be higher than accuracy (overconfident). If it is with soft label, the largest probability is pushed to approach to the value of the correct soft label.).

If the hard labels already lead the model to be underconfident (the largest probability is smaller than accuracy), then after distillation (equivalent training with soft labels), student becomes more underconfident and more miscalibrated than teacher, as shown in Fig. 4.3. The above reason also explains the right plot in Fig. 4.2, that asαincreases, teacher-provided labels become softer, student's ECE goes up. But if the hard labels lead the model to be overconfident, then after distillation, student intuitively could get trained to prevent overconfidence and be more calibrated.

A simple experiment is done and can be viewed in the following. We choose an overconfident model (one-layer NN on Fashion-MNIST dataset[2] [60]) and

[2]Fashion-MNIST is a Zalando's article images dataset, with $60,000$ training samples and $10,000$ test samples. Each sample is a Zalando image, with a label from 10 classes.

an underconfident model (GCN on Pubmed). Here our KD experiments are conducted with students getting trained solely with teacher-produced soft labels, i.e. $\lambda = 1$ in Eq. 4.3. Results verify Property 1, as shown clearly in Table 4.3 and Fig. 4.4. After a first distillation from the teacher, the student becomes more miscalibrated for the underconfident model and more calibrated for the overconfident model. After a second distillation from the student, the student's student becomes more miscalibrated for the underconfident model and more calibrated for the overconfident model than the student.

Table 4.3: Two examples of KD's effects on over/under-confident models.

	Dataset	Model	Teacher ECE	Student ECE	Student's student ECE
Overconfident	Fashion-MNIST	1layerNN	5.4	2.8	2.3
Underconfident	Pubmed	GCN	7.8	19.3	24.8

4.4 Conclusion

As stated in Claim 1, if the teacher is overconfident, the student trained with KD could become more calibrated. If the teacher is underconfident, the student becomes more miscalibrated.

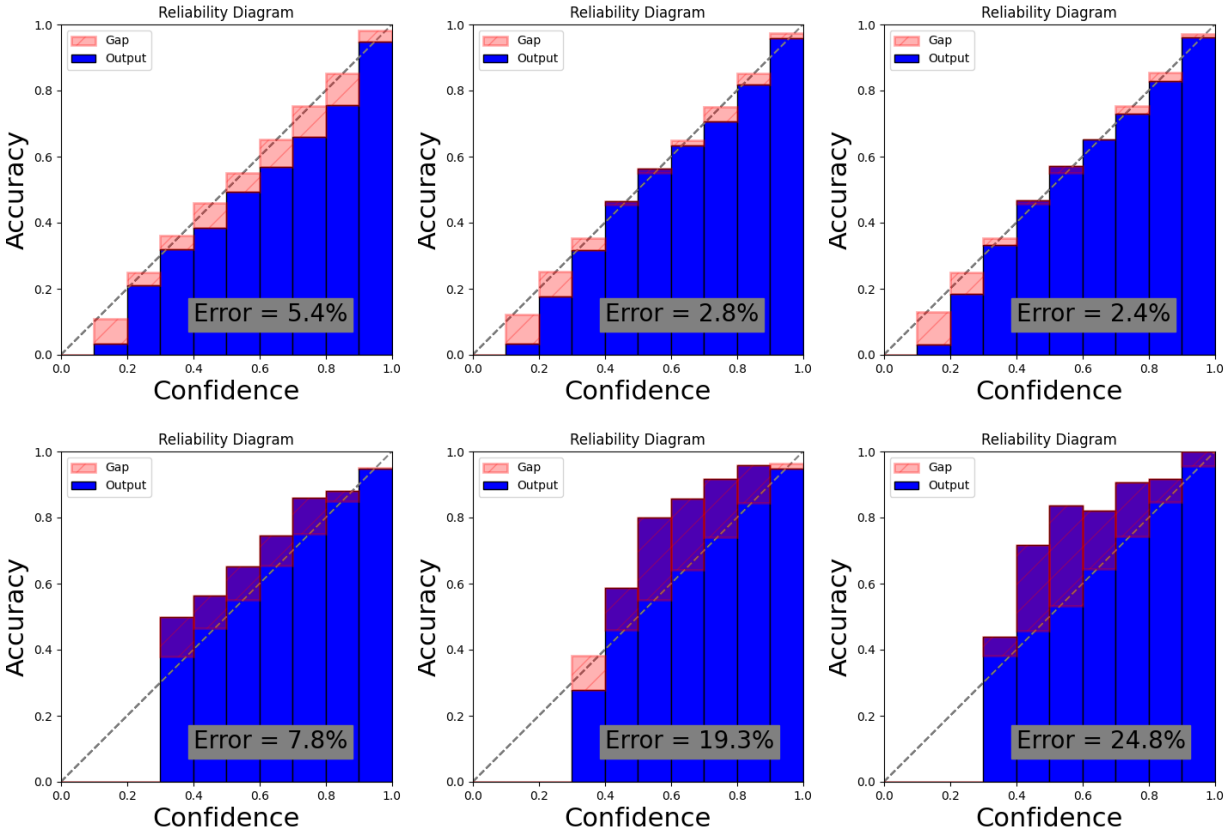


Figure 4.4: Two examples of KD's effects on over/under-confident models. Top: One-layer NN on Fashion-MNIST. Down: GCN on Pubmed. From left to right: teacher, student, student's student.

Chapter 5

Graph signal processing for calibration

In Chapter 4, we analyze the relationship between KD and calibration. In this chapter, we dive deeper into the theory of graph networks from graph signal processing. We first introduce graph signal processing and spectral graph models and then study the calibration from this perspective.

5.1 Introduction

There are close connections between graph signal processing and graph neural networks. From the first benchmark GNN model, i.e., GCN [21], people started to understand and improve graph networks from the perspective of graph signal processing [10, 22, 61, 62]. In this section, we will give a detailed introduction. It covers understanding graph signals, message passing, graph convolutions, over-smoothing, benchmark spectral GNN, and low-pass filtering. The final goal of the introduction is to understand the nature of spectral GNNs is low-pass filtering.

5.1.1 From time signals to graph signals

Following Ref. [1], there is a connection between discrete time-varying signals and graph signals. Suppose we have N scalar signals f_i[1] ($i \in \{1, .., N\}$), the time-varying signals are represented as $\{f(t_0), f(t_1), ..., f(t_{N-1})\}$, where

[1] In the real task, this signals corresponds to the feature matrix X.

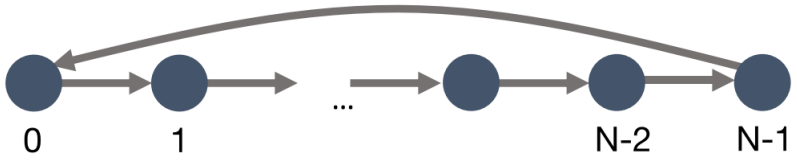


Figure 5.1: Representation of a (cyclic) time-series as a chain graph [1].

t_i denotes the time varying, we view them on a cyclic chain graph, like in Fig. 5.1. Therefore $f[(t+1)_{mod\ N}]$ represent the next time signal of $f[t]$. The adjacency matrix $A_c \in \mathbb{R}^{N\times N}$ and Laplacian[2] $L_c \in \mathbb{R}^{N\times N}$ of the one-dimension graph correspond to:

$$\begin{aligned} A_c[i,j] &= \begin{cases} 1\ if\ i\ and\ j\ are\ neighbors \\ 0\ otherwise, \end{cases} \\ L_c &= I - A_c. \end{aligned} \tag{5.1}$$

Therefore equivalently we can represent time shifts as multiplications by the adjacency matrix and represent the difference operation by the multiplication by the Laplacian:

$$\begin{aligned} (A_c f)[t] &= f[(t+1)_{mod\ N}] \\ (L_c f)[t] &= f[t] - f[(t+1)_{mod\ N}]. \end{aligned} \tag{5.2}$$

5.1.2 From convolution filter to message passing

From Section 5.1.1, we see the connection between the adjacency and Laplacian of a graph and the notions of shifts and differences of a signal. Following Ref. [1], similarly, we can represent convolution[3] of a graph (represented by a convolution filter h) by matrix multiplication on the signal vector $f \in \mathbb{R}^N$:

$$\begin{aligned} (f \star h)(t) &= \sum_{\tau=1}^{N-1} f(\tau) h(t-\tau) \\ &= Q_h f, \end{aligned} \tag{5.3}$$

[2]Definition in Chapter 2.

[3]Let f and g be two functions of x. The general convolution operation $\star$ is defined as $(f \star h)(x) = \int f(y)h(x-y)dy$.

where $Q_h \in \mathbb{R}^{N\times N}$ is the matrix version of convolution filter h, and $f = [f(t_0), f(t_1), ... f(t_{N-1})]^T \in \mathbb{R}^N$ is the vector representation of signal f. We can see that in order to satisfy the second equality of the above fomular, Q_h needs to satisfy *translation equivariance*, or to say, to commute with adjacency matrix A_c:

$$A_c Q_h = Q_h A_c, \tag{5.4}$$

which is equivalent to:

$$Q_h = \sum_{i=0}^{N-1} \alpha_i (A_c)^i, \tag{5.5}$$

where α_i is an arbitrary constant. The first-order expansion corresponds to the message passing formula in the basic GNNs (see Eq. 2.4):

$$Q_h = I + A. \tag{5.6}$$

In this way, we expand the convolution matrix Q_h to the message passing formula. This means there is connection between convolution on a graph and message passing in GNNs.
For a general graph, Laplacian $L = D - A$. But in practice, people usually use *symmetric normalized Laplacian* $L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$ and *symmetric normalized adjacency matrix* $A_{sym} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. L_{sym} and A_{sym} satisfy $L_{sym} + A_{sym} = I$.

5.1.3 Representing graph convolutions by polynomials of Laplacian

Following Ref. [1], consider the eigen decomposition of the general graph Laplacian L:

$$L = U \Lambda U^T, \tag{5.7}$$

where the eigenvector U is defined as *graph Fourier modes*, and eigenvalues of the diagonal matrix Λ is defined as *frequency of the graph*. Thus, for a signal $f \in \mathbb{R}^{|\mathcal{V}|}$ ($N = |\mathcal{V}|$) defined on vertex of the graph, we have the Fourier transform and its inverse transform:

$$\begin{aligned} f^{'} &= U^T f \\ f &= U f^{'} \end{aligned} \tag{5.8}$$

With the defined Fourier modes, we can rewrite a graph convolution as:

$$f \star h = U(U^T f \circ U^T h) = (U diag(\alpha_h) U^T) f, \tag{5.9}$$

where U is the matrix of eigenvectors of Laplacian and $\circ$ denotes element-wise products. $diag(\alpha_h)$ is a matrix with value α_h on the diagonal, with $\alpha_h = U^T h \in \mathbb{R}^{|\mathcal{V}|}$ called *graph Fourier coefficient*. α_h is the only parameter here. To ensure that it corresponds to a meaningful convolution on the graph, a natural solution is to *parameterize* α_h *based on the eigenvalues* Λ *of the graph*, such as approximating it as a degree N polynomial $p_N(\Lambda) = \sum_{i=0}^{N} \theta_i \Lambda^i$, with coefficients θ_i, therefore:

$$\begin{aligned} f \star h \approx (U p_N(\Lambda) U^T) f &= U(\sum_{i=0}^{N} \theta_i \Lambda^i) U^T f \\ &= p_N(L) f, \end{aligned} \tag{5.10}$$

where $p_N(L)$ is a N polynomial of $L^i = U L^i U^T$. Till now this is the key idea that graph convolutions are represented by polynomials of Laplacians.

5.1.4 Understanding over-smoothing from graph signal processing

In Section 2.1.3, we discuss the over-smoothing. Recall that the intuitive of over-smoothing is that after too many rounds of message passing, the embedding of nodes becomes similar and uninformative.
Suppose we have a simple basic GNN (where we remove the non-linearity and self loops for simplicity) as in Section 2.1.3:

$$H^{(k)} = A_{sym} H^{(k-1)} W^{(k)}, \tag{5.11}$$

where A_{sym}, $H^{(k-1)}$ and $W^{(k)}$ are symmetric normalized adjacency matrix, $(k-1)$-th layer hidden embedding and k-th layer weight matrix. Then after K rounds updatings, we will end up with a $K-th$ power of adjacency matrix and weight matrix:

$$H^{(K)} = A_{sym}^K H^{(0)} W = A_{sym}^K X W, \tag{5.12}$$

where X is the node feature matrix and $W = W^1 W^2 ... W^K$.

A higher order of K in A_{sym}^K emphasizes the largest eigenvalue of A_{sym} (this is by the eigenvalue composition of A_{sym}). And since $L_{sym} + A_{sym} = I$, the largest eigenvalue of A_{sym} correspond to the smallest eigenvalues of L_{sym}. Thus, multiplying a signal by high powers of A_{sym}, $A_{sym}^K X$, corresponds to a convolutional filter based on its lowest eigenvalues (or frequencies) of L_{sym}. Therefore, we can interpret $A_{sym}^K X$ as a convolutional filter based on the lowest-frequency signals of graph Laplacian. A result of stacking many rounds of message passing leads to signals with a low-pass filter for many rounds, and in the worst case, these node representations *converge to constant values*.

5.1.5 Understanding GCN from graph signal processing

We now analyze GCN and show that it is closely related to graph signal processing.

Graph Convolutional Networks (GCN) [21] employs a first-order approximation (K = 1) of Eq. (5.10) with coefficients $\theta_0 = 2\theta$ and $\theta_1 = -\theta$ and attain the basic GCN convolution formular:

$$\begin{aligned} f \star h &\approx \theta(I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) f \\ &= \theta(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}) f, \end{aligned} \tag{5.13}$$

where the second step of the above equation is *renormalization trick*: $I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, with $\tilde{A} = A + I$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.
This is similar with but not same-derived with the formula in a spatial perspective (see Eq. 2.13) $H^k = \sigma(\tilde{A} H^{k-1} W^k)$, where W^k is a learnable parametric matrix.
Following Ref.[10], the GCN filter in (5.13) corresponds to the a matrix $S_{1-order}$ so called *propagation matrix*. The $1 - order$ here denotes it is a first-order approximation.

$$S_{1-order} = I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}. \tag{5.14}$$

Since the normalized Laplacian is $L_{sym} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, then

$$S_{1-order} = 2I - L_{sym}. \tag{5.15}$$

In the eigen decomposition of Laplacian (see Eq. 5.7), the power feature propagation matrix $S_{1-order}$ corresponds to filter coefficients α_i ($i \in [0, |\mathcal{V}|]$) (see Eq. (5.9)):

$$\alpha_i = 2 - \lambda_i, \tag{5.16}$$

where λ_i denotes the eigenvalues of L_{sym}, and $\lambda_i \in [0, 2]$ in spectral range. This α_i represents the topology information of the node i in the graph. This topology information participates in the convolution with the feature matrix X as $(Udiag(\alpha_i)U^T)X$ (see Eq. (5.9)). For a K layer convolution layer, the filter coefficient for node i is $\alpha_i^K = (2 - \lambda_i)^K$. Fig. 5.2 shows an example of features and coefficients α_i^K in Cora dataset[49]. Before filtering, features are relatively small around 0. After filtering. the scaled spectrum for $K > 1$ allows a *low-pass filtering*, to amplify frequencies $\lambda_i < 1$.

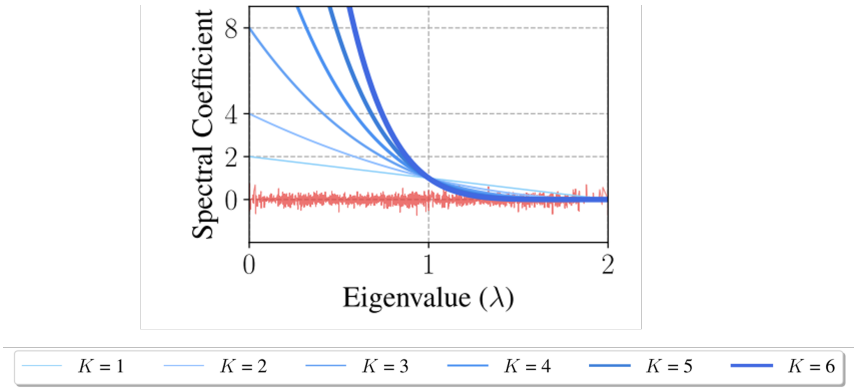


Figure 5.2: Features in Cora in the spectral range (red) and corresponding filter coefficients (blue). [10]

5.1.6 Understanding low-pass filtering

With the above intuition and visual low-pass filtering, following Ref. [22], we introduce the assumption of low-pass filtering in GNNs.

Assumption 5.1 [22]. *In graph networks, observed features X consist of low-frequency true features $\bar{X}$ and high-frequency noise Z. $\bar{X}$ has a frequency at most $0 \leq \epsilon \ll 1$ and noise Z follows a Gaussian distribution. True features are informative enough for the machine learning task.*

This assumption does some assumption to the function of features in GNNs: $X = \bar{X} + Z$. Further, if we only have true features $\bar{X}$ without any structure

information about the graph, it is also enough to be trained by the model for the machine learning task. Further we have the following theorem.

Theorem 5.2[4] [22]. *Under the Assumption 1, the outputs of GNNs are similar to the outputs of using a NN to true features $\bar{X}$.*

This theorem shows that with input node feature matrix X, the function of GNNs is to have a low-pass filtering of X, such that obtained $\hat{X}$ could approximate the true features $\bar{X}$. Take an example of SGC, $H = \sigma(\tilde{A}^K XW$ (see Eq. 2.15). There are two steps, incorporating the structure information $\hat{X} = \tilde{A}^K X$ and doing a equivalent perceptron $H = \sigma(\hat{X}W)$ with input $\hat{X}$. $\hat{X}$ is a feature matrix approximating true features $\bar{X}$ by multiplying with adjacency matrix $\tilde{A}$. Actually, as Lemma 5 and Corollary 6 in [22] shown, the difference between approximated features $\hat{X}$ are accurate estimations of true features $\bar{X}$: $\hat{X}$ and $\bar{X}$ have an upper bound, and this upper bound could be relatively small by a specific choice of K. We also verify this theorem in Section 5.2.1.

5.2 Analysis on calibration

When analyzing calibration performance from the perspective of graph signal processing, several first questions to ask are what is the influence of our eigenvalues λ_i of Laplacian on calibration here? Can we reconstruct a low-pass signal that outputs also low calibration like Theorem 5.2? Our belief is that by choosing low frequencies λ_i, graph networks can obtain better accuracy but almost the same calibration performance as the original models.

Recall the equivalent weak version of Theorem 2.2 (Eq. 2.32): with a post-processing calibration method g, an softmax function q, input samples X and true labels K, a model f is calibrated if $\underset{g:\mathbb{R}^m \to \mathbb{R}^m}{argmin}\, L_2(q \circ g \circ f, X, K) = id$.

Theorem 2.2 tells us that a model is perfectly calibrated if no post-processing methods could help calibrate anymore. From this perspective, we consider that the calibration information is hidden in the model: *post-processing methods' effect is to distill this information.* We have the following claim.

[4] For a formal version, see Theorem 7 and 8 in [22].

Claim 5.1. *Under Assumption 5.1, true signals cannot help obtain better calibration performance by improving hidden calibration information in graph networks.*

Claim 5.1 states by applying low-pass filtering to signals to filter true signals, the calibration of models might be better or worse than without filtering, but after applying post-processing calibration methods, calibrated performance retains at the same level as directly applying post-processing calibration methods without filtering.

5.2.1 Empirical evidence of Claim 5.1

Experiment process. We first verify Claim 5.1 in citation datasets: Cora, Citeseer, and Pubmed[49]. We follow the process of Nt et al. [22] to compute k-frequency component[5]: 1. Compute the graph Fourier modes U from Laplacian $\tilde{L}$; 2. Add Gaussian noise to the input features; 3. Compute the first k frequency components $\hat{X}_{:k} = U[:k]^T \tilde{D}^{1/2} X$, where $U[:k]$ represents the first k rows of matrix U; 4. Reconstruct features: $\tilde{X}_k = U[:k]\hat{X}_{:k}$; 5. Train with a simple two-layer multi-layer perceptron (MLP); 6. Distill the calibration information from the model using a post-processing method, we choose a representative calibration methods, histogram binning [41] (see Chapter 2).

Experiment setting. The hidden layer of MLP is set to 32, with a ReLU activation function. Optimizer is Adam with lr of 0.01 and weight decay of $5e-4$. Epochs are set to 200 with early-stopping of patience 10. Results are over 10 independent runs.

Results and discussions. We can see the following results from Fig. 5.3 and Fig. 5.4.

1. Inputting low-frequency signals could help improve model accuracy, especially in Cora and Citeseer datasets, as shown in the first row in Fig. 5.3. Fraction of high-frequency components signals performs like adding noise, in terms of accuracy. This verifies Theorem 5.2.

2. Low-frequency signals cannot help calibration. Uncalibrated calibration performance is more unstable than accuracy as frequency components in-

[5]This is different with Nt et al.'s setting. In their paper, the first k frequency components is computed as $\hat{X}_{:k} = U[:k]^T X$. Reconstructed features are computed as $\tilde{X}_k = \tilde{D}^{-1/2} U[:k]\hat{X}_{:k}$.

crease as shown in the second row in Fig. 5.3. There is no obvious empirical relationship between them. From confidence histograms in Fig. 5.4 we observe that on a higher frequency, scores tend to shift to the left and assemble in small-confidence bins. This does not alleviate the underconfidence.
3. Adding noise to signals makes uncalibrated ECE more unstable, in some cases adding noises could even decrease ECE, when we compare the different noise level in Fig. 5.3.
4. Post-processing calibration methods have denoising abilities. After histogram binning, almost all frequency components show ECE at the same level, as shown in the last row in Fig. 5.3. Such post-processing methods are simple but effective.

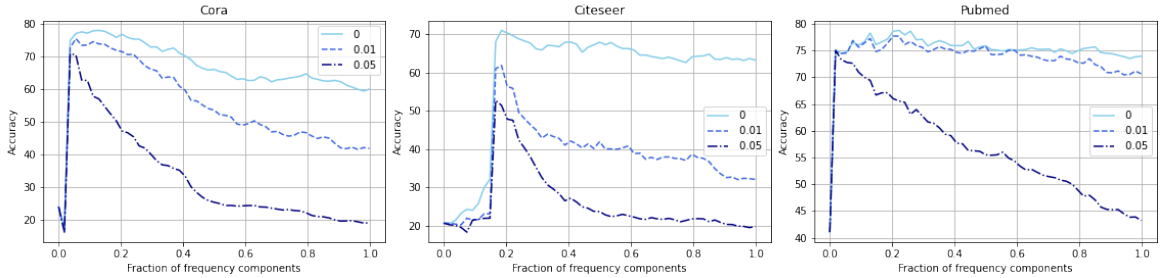
5.2.2 Spectral low-pass filtering calibration performance

In above we analyze the influence of frequency components on a MLP model. We further analyze the influence of frequency components on three representative spectral graph networks (SGC [10], GCN [21], and gfNN [22]).
Setting. We run experiments the influence of frequency components on accuracy, uncalibrated ECE, calibrated ECE after histogram binning and temperature scaling using models SGC, GCN and gfNN on Cora[49]. Parameters are kept the same as Section 3.1.3. Results are over 10 independent runs
Results and discussions. In Fig. 5.5 we show the results. Compared the results of accuracy and uncalibrated ECE plots in the first and second rows in Fig. 5.4 and Fig.5.5, GCN, SGC and gfNN all attain much more stable performance when increasing frequency components. As we expected, spectral graph networks have an effect of low-pass filtering for both accuracy and calibration.

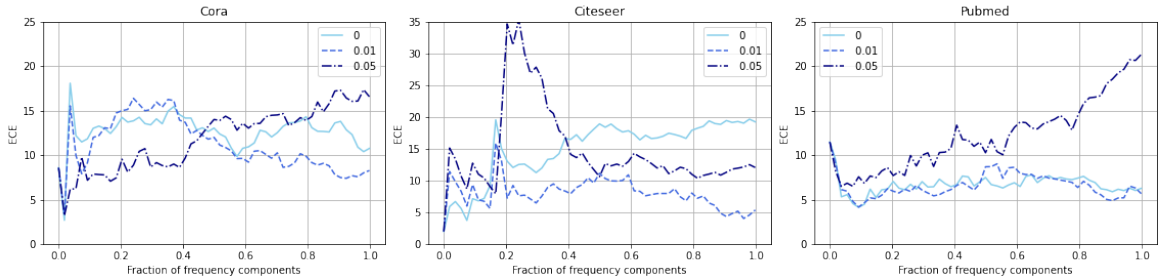
5.3 Conclusion

Although graph signal processing plays an important role in gaining meaningful information to classification tasks, it is not a dominative factor to model calibration performance. The nature of GNNs, i.e. low-pass filtering, could not increase the hidden calibration information in models. Viewing

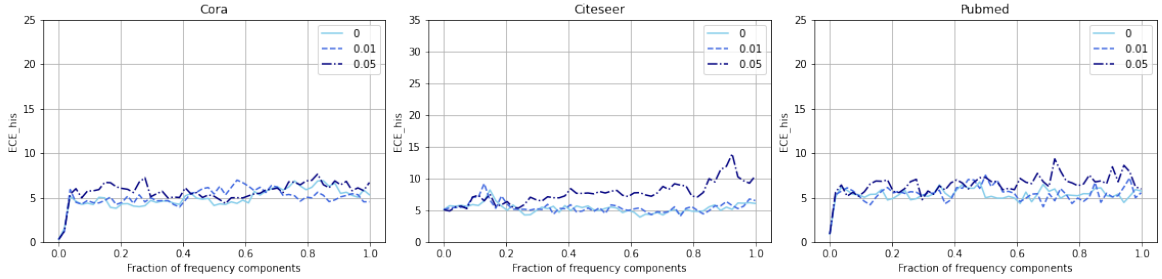
high-frequency components of signals as noises, is not suitable for calibration. They are not "real" noises for calibration. Besides, post-processing calibration methods have the ability to "denoise". Besides, spectral GNNs, like GCN, have the effects of low-pass filtering for both accuracy and calibration.



(b) Accuracy



(e) Uncal. ECE



(h) ECE after Histogram Binning

Figure 5.3: Model performance on citation datasets. The first, second and third column is the result on Cora, Citeseer and Pubmed respectively. The first, second, third row is the result withrespect to accuracy, uncalibrated ECE, ECE after histogram binning respectively Gaussian noises are added in the level of $\{0, 0.01, 0.05\}$.

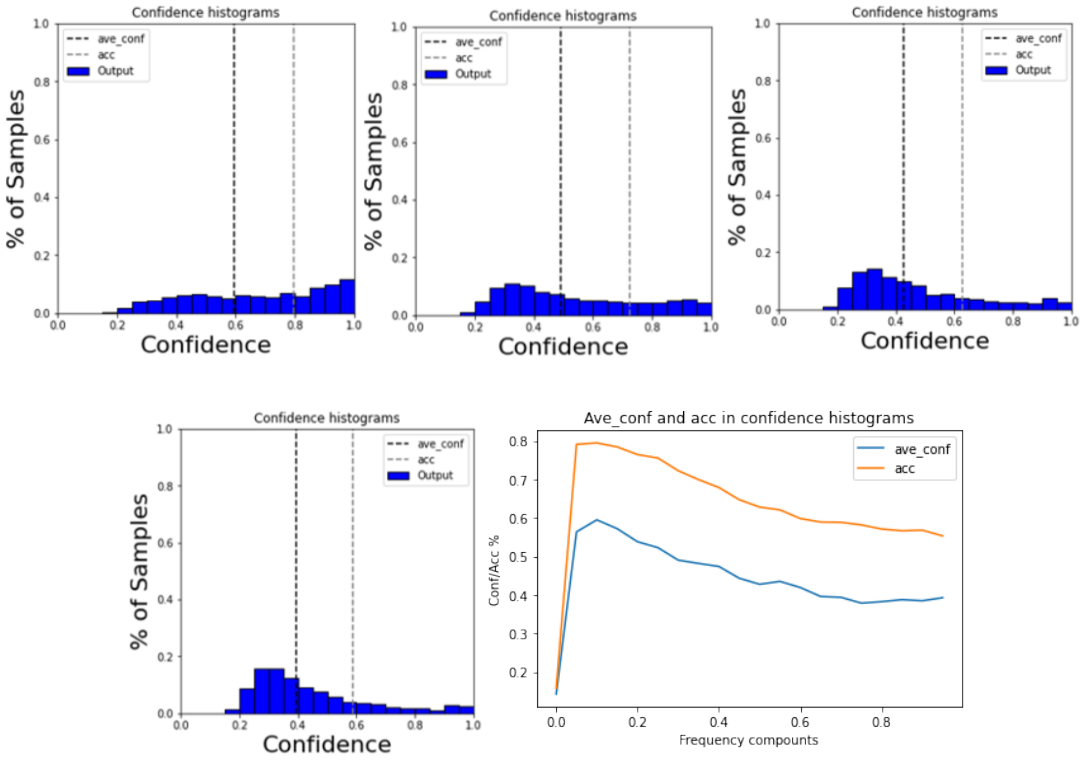
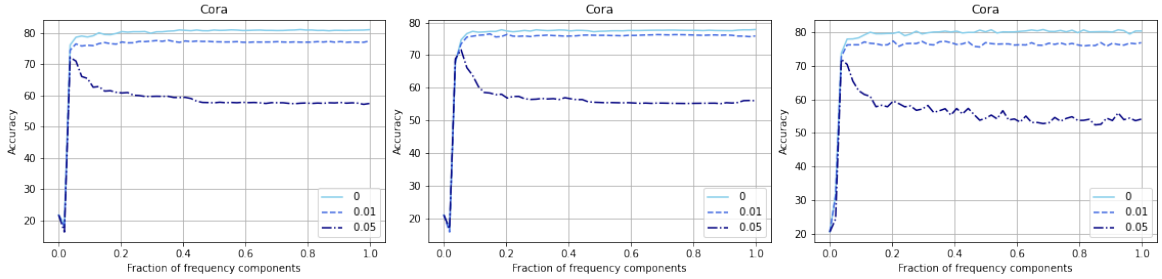
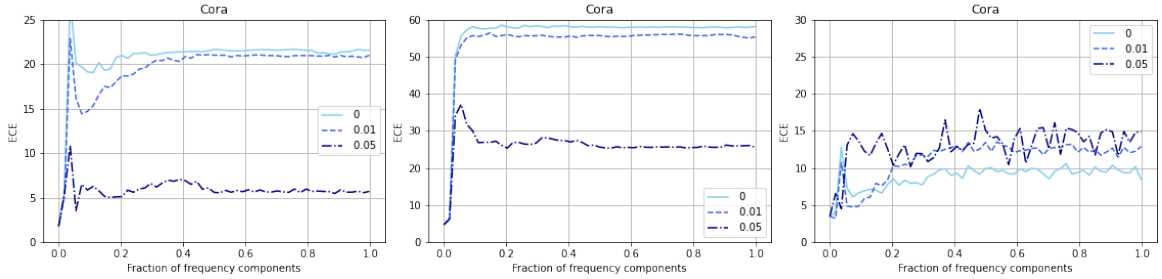


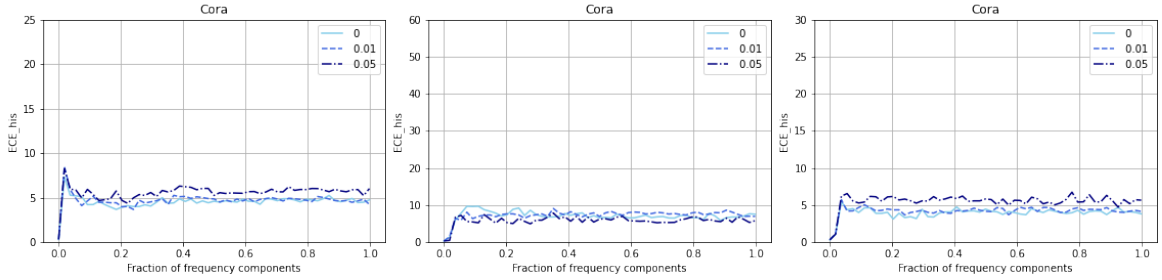
Figure 5.4: A little example of averaged confidence and accuracy on frequency components with a MLP on Cora dataset. The first four plots correspond to fraction of frequency components of 0.1, 0.3, 0.5 and 0.7 respectively. The last plot is the plot of averaged confidence and accuracy as frequency component from 0 to 1. Both confidence and accuracy decrease with accuracy. This may explain why ECE becomes unstable and even increasing as the fraction of frequency components increases.



(b) Accuracy



(e) Uncal. ECE



(h) ECE after Histogram Binning

Figure 5.5: Model performance on Cora. The first, second and third column is the result of models GCN, SGC and gfNN respectively. The first, second, third row is the result with respect to accuracy, uncalibrated ECE, ECE after histogram binning respectively.

Chapter 6

Modifying loss function: adding calibration term

Since the requirement of a calibrated network is different from the requirement to find the minimum of the cross-entropy loss function, it is hard to come up with a built-in method to satisfy the minima of two different loss functions at the same time, unless we replace the cross-entropy with a loss function that has both cross-entropy and calibration loss effects. Adding calibration loss to the cross-entropy loss could be one possible solution.

6.1 Expected calibration loss

A first idea is to add a calibration error term. This is inspired by one loss term from Tomani et al. [46]. Given the original cross-entropy loss L_{CE}, the new loss is defined as:

$$\begin{aligned} L &= \alpha L_{CE} + (1-\alpha) L_{cal} \\ L_{cal} &= \sqrt{\sum_{i=1}^{n} (acc(B_{m_i}) - \hat{p}_i)^2}, \end{aligned} \tag{6.1}$$

where B_{m_i} denotes the bin that sample i is in.

Setting. Following the setting from Tomani et al. [46], in the experiment, we set L_{cal} to anneal with epochs, since the early epochs are usually used

Table 6.1: (Uncalibrated performance) Calibration of GCN and GAT with respect to accuracy, ECE, and Marginal-ECE (mean±standard deviation over 10 independent runs). The best results when comparing between two loss functions are displayed in bold.

Dataset	Model	Acc. (L_{CE})	ECE (L_{CE})	M-ECE (L_{CE})	Acc. (L)	ECE (L)	M-ECE (L)
Cora	GCN	81.34±0.47	21.17±0.73	6.50±0.16	**81.49±0.48**	**19.27±0.72**	**5.97±0.19**
	GAT	**83.26±0.41**	15.19±0.54	5.08±0.20	83.24±0.40	**13.15±0.60**	**4.53±0.20**
Citeseer	GCN	**71.24±0.65**	21.04±0.89	8.49±0.25	70.95±1.15	**12.63±1.64**	**6.05±0.45**
	GAT	70.90±0.56	17.00±0.55	7.61±0.27	**71.00±0.66**	**4.45±0.51**	**4.56±0.47**
Pubmed	GCN	**79.17±0.46**	6.59±0.65	5.30±0.51	78.68±0.44	**4.47±0.62**	**4.67±0.75**
	GAT	78.22±0.41	4.46±0.76	4.70±0.75	**78.23±0.38**	**4.30±0.90**	**4.63±0.66**

for reaching the cross entropy minimum. More precisely,

$$\begin{aligned} anneal\_coef &= \min\left(1, \frac{epoch}{EPOCHS \cdot max}\right) \cdot \lambda \\ \hat{L}_{cal} &= anneal\_coef \cdot L_{cal} \\ L &= \alpha L_{CE} + (1-\alpha)\hat{L}_{cal}, \end{aligned} \tag{6.2}$$

where *epoch* and $EPOCHS$ are the current training epoch and the total epochs number, max and λ being two parameters.
In experiment, we tune α, max and λ on validation set from $\{0.9, 0.95\}$, $\{1e-2, 2\}$ and $\{1e-2, 5, 10, 80, 160\}$. In Table 6.2 we list the chosen hyperparameters. We fix two options, $\alpha = 0.95, max = 1e-2$, $\lambda = 5$ or $\alpha = 0.95, max = 2, \lambda = 160$ for cases except GAT on Pubmed. Experiments are run 10 times.
Results. In Table 6.1 we list the results of GCN and GAT on citation datasets. The new loss function L could help improve calibration while keeping the accuracy at the same level.
There is also options to add other calibration loss functions to L_{CE}, e.g. marginal calibration error, or KS-error, described in Section 2.2.2. We believe these loss functions also could help calibrate.

6.2 Conclusion

Adding a simple expected calibration loss term to the cross-entropy could help improve calibration while keeping the accuracy at the same level for

Table 6.2: Hyperparameter choice.

Dataset	Model	alpha	max	lambda
Cora	GCN	0.95	1e-2	5
Cora	GAT	0.95	1e-2	5
Citeseer	GCN	0.95	2	160
Citeseer	GAT	0.95	2	160
Pubmed	GCN	0.95	2	160
Pubmed	GAT	0.95	1e-2	1e-2

GNNs. Adding other calibration loss functions could also be an option.

Chapter 7

Ratio-binned scaling

7.1 Motivation

Results in Chapter 3 show that GNNs tend to exhibit underconfidence on citation datasets. However, two further questions could be asked. First, even though the overall model is underconfident, is it possible to have some node-level differences? E.g., some nodes' predictions tend to be overconfident while the others are underconfident. Secondly, since for graph data, topology plays an important role in the node classification result, e.g., in the homophilic graph, a node with same-class neighbors has a high probability to be in the same class. Then does this topology information also have an influence to calibration, i.e., a node's neighbors also contribute to the confidence of the node's prediction?

7.2 Methods

To explore the above questions, a simple metric, *same-class-neighbor ratio*, is calculated. As shown in Fig. 7.1, suppose node i has neighbors all being of the same class while node j only has $1/3$ proportion same class neighbors. If node i and j have the same predicted label, then intuitively, node i should be more confident about its prediction, i.e. $\hat{p}_i > \hat{p}_j$. But how about the confidence's comparison with accuracy?

Claim 1. *In graph neural networks on homophilic graphs, nodes with higher same-class-neighbor ratios tend to have more confident predictions than lower ratios'. Besides, nodes with higher ratios tend to have overconfi-*

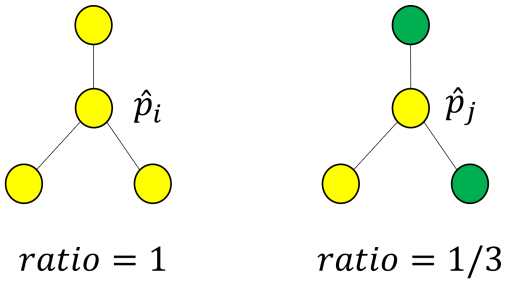


Figure 7.1: The illustration of nodes of different same-class-neighbor ratio. Yellow and green denote different classes.

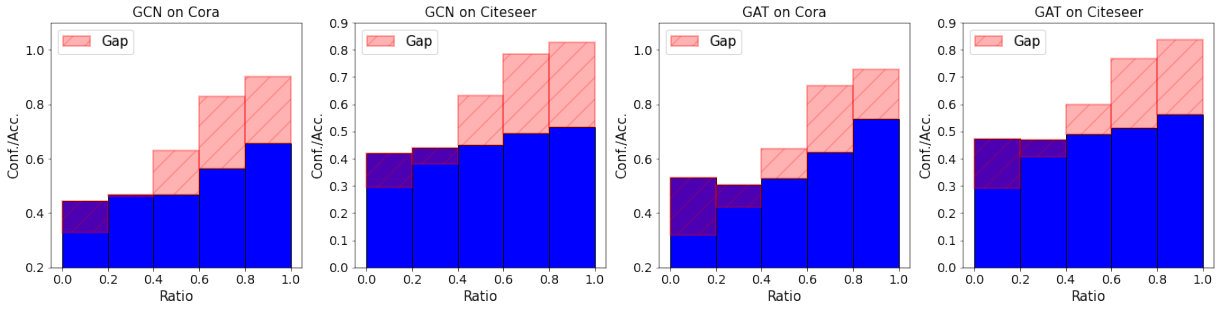


Figure 7.2: Ratio-based reliability diagrams for GCN and GAT on Cora and Citeseer.

dent predictions, while nodes with lower same-class-neighbor ratios tend to have underconfident predictions.

Our empirical experiment evidences Claim 1, illustrated in Fig. 7.2. Here in order to explore the relationship between same-calss-neighbor ratios and the confidence of nodes, we calculate each node's same-calss-neighbor ratio using its ground-truth label. Then we group them into 5 equal-space interval bins according to their ratios. Similar to reliability diagrams [30, 31], we draw the averaged confidence output of nodes in each bin as blue bars and the accuracy in the bin as red bars. A clear observation is that a node tends to gain higher confidence with higher proportion of same-class neighbors around it. While at the same time, bins with higher ratios tend to exhibit underconfidence and bins with lower ratios tend to exhibit overconfidence. The ratio could present to be an indicator to discriminate underconfident and overconfident bins. Therefore, consequent calibration could be implied to samples separately depending on underconfidence or overconfidence.

However, in practice, due to semi-supervised setting, we do not know labels

of most nodes in a graph, therefore, cannot calculate this ratio. One natural option is to replace nodes' ground-truth labels with their confidences then calculate the aggregated probabilities. More precisely, for node i, the predicted ratio is calculated as:

$$\hat{r}(i) = \frac{1}{|N(i)|} \sum_{j \in N(i)} f(X_j)_{\hat{y}_i}, \tag{7.1}$$

where $f(X_j)$ denotes the K-length probability vector for node j and $f(X_j)_{\hat{y}_i}$ denotes the probability of class $\hat{y}_i$ for node j, with $\hat{y}_i$ being the predicted label for node i.

7.2.1 Ratio-binned scaling calibration

We propose our method Ratio-Binned Scaling calibration (RBS). RBS first calculates the predicted same-class-neighbor ratio according to the output of GNNs, then group nodes into M bins according to the ratio, and finally applies a temperature scaling [16] in each bin.
More precisely, let f be a trained GNN model, with $f(X_i)_k = \sigma(z_i)_k$ being the k-class probability for node i with z_i being the logit vector for node i and σ being the softmax function. Let $\mathcal{V}_v$ and $\mathcal{V}_t$ denote the set of validation samples and test samples respectively. A ratio $\hat{r}(i)$ is calculated for each node i, with $i = \{1, ..., N\}$. A set of equal-space bins for validation set $\{B_1^v, B_2^v, ..., B_M^v\}$ and test set $\{B_1^t, B_2^t, ..., B_M^t\}$ are calculated and used to partition nodes in $\mathcal{V}_v$ and $\mathcal{V}_t$ respectively. M separate temperatures $\{T_m\}$ ($m = \{1, 2, ..., M\}$) are learnt by minimizing the NLL of nodes in the bin of $\{B_1^v, B_2^v, ..., B_M^v\}$. Then for node j in B_m^t, calibrated probability $\hat{q}_j = \sigma(z_j/T_m)$.

7.3 Experiments, results and discussions

Setting. In experiments, we apply our proposed post-processing calibration method (RBS) to GCN [21] and GAT [25] on Cora, Citeseer and Pubmed datasets [49]. All parameters are the same as experiments in Section 3.1.4. Bin number M in RBS are tuned in $\{2, 3, 4, 5\}$. Results are compared with post-processing methods in Section 3.1.4 (see Table.3.3).

Table 7.1: Calibrated performance with respect to ECE (mean±standard deviation over 100 independent runs). The best GNN model for each calibration method is underlined. The best calibration method for each GNN model is displayed in bold.

Dataset	Model	Uncal.	His. bin.	Iso. reg.	BBQ	Tem. scal.	Meta-Cal	RBS
Cora	GCN	21.17±0.73	4.42±0.77	3.92±0.71	4.50±0.73	3.73±0.62	3.88±0.60	**3.69±0.70**
	GAT	15.19±0.54	4.83±0.56	4.07±0.57	4.12±0.55	3.75±0.73	**3.37±0.65**	3.54±0.76
	SGC	24.95±0.19	4.89±0.30	3.63±0.27	4.25±0.43	3.95±0.18	4.02±0.40	**3.23±0.11**
	gfNN	16.30±0.35	3.18±0.59	**3.08±0.43**	3.74±0.45	3.22±0.49	3.53±0.45	3.57±0.79
	APPNP	16.96±0.56	4.37±0.67	3.78±0.59	3.99±0.69	3.38±0.59	3.67±0.66	**3.26±0.64**
Citeseer	GCN	21.04±0.89	**4.58±0.76**	4.62±0.92	5.44±1.00	4.87±0.67	4.93±0.74	5.15±0.84
	GAT	17.00±0.55	5.32±0.75	5.19±0.68	5.27±0.73	5.88±0.64	6.27±0.65	**4.31±0.72**
	SGC	42.89±0.08	4.63±0.21	3.94±0.15	5.94±0.31	4.47±0.15	4.50±0.23	**3.07±0.14**
	gfNN	20.06±0.54	**4.40±0.66**	4.52±0.75	4.55±0.70	5.11±0.49	4.76±0.56	5.49±1.54
	APPNP	13.24±0.57	4.96±0.92	**4.85±0.84**	5.47±0.87	5.00±0.78	5.40±0.73	5.26±0.82
Pubmed	GCN	6.59±0.65	5.06±0.80	4.85±0.74	4.98±0.83	**4.38±0.60**	5.03±0.79	4.49±0.60
	GAT	4.46±0.76	4.77±0.86	4.73±0.77	4.97±0.86	3.96±0.69	4.56±1.09	**3.70±0.65**
	SGC	20.99±0.02	4.43±0.20	**3.74±0.29**	4.00±0.21	4.46±0.19	4.41±0.69	4.24±0.14
	gfNN	4.38±0.42	5.35±0.82	4.98±0.65	4.74±0.67	4.83±0.52	6.18±0.82	**4.71±1.33**
	APPNP	5.14±0.83	4.90±0.77	4.47±0.60	4.74±0.78	4.34±0.75	5.11±0.95	**4.21±0.75**

Results and discussions. Results are shown in Table. 7.1. With the best result for each GNN model displayed in bold, we can see RBS gains the best calibration performance in 8 out of 15 experiments and outperforms temperature scaling in 10 out of 15 experiments.

To further evidence the effectiveness of the intuition of RBS, we show the calibration results of real-ratio-binned scaling (RRBS), where we assume that we know the ground-truth ratios of nodes. Although this information is not available in practice, this result further evidences the effectiveness of the intuition of our method. Results are shown in table 7.2. RRBS outperforms SOTA in 12 out of 15 experiments, and outperforms RBS in 14 out of 15 experiments.

Table 7.2: Calibrated performance for SOTA, RBS and RRBS (SOTA denotes the best one result from histogram binning, isotonic regression, BBQ, temperature scaling and meta calibration) with respect to ECE (mean±standard deviation over 100 independent runs). The best calibration method for each GNN model is displayed in bold.

Dataset	Model	Uncal.	SOTA	RBS	RRBS
Cora	GCN	21.17±0.73	3.73±0.62	3.69±0.70	**3.19±0.57**
	GAT	15.19±0.54	**3.37±0.65**	3.54±0.76	3.46±0.62
	SGC	24.95±0.19	3.63±0.27	3.23±0.11	**2.86±0.16**
	gfNN	16.30±0.35	**3.08±0.43**	3.57±0.79	3.36±0.48
	APPNP	16.96±0.56	3.38±0.59	3.26±0.64	**3.15±0.51**
Citeseer	GCN	21.04±0.89	4.58±0.76	5.15±0.84	**3.98±0.69**
	GAT	17.00±0.55	5.19±0.68	**4.31±0.72**	4.85±0.86
	SGC	42.89±0.08	3.94±0.15	3.07±0.14	**2.88±0.09**
	gfNN	20.06±0.54	4.85±0.84	5.26±0.82	**4.64±1.23**
	APPNP	13.24±0.57	4.85±0.84	5.26±0.82	**3.8±0.73**
Pubmed	GCN	26.59±0.65	4.38±0.60	4.49±0.60	**3.34±0.65**
	GAT	4.46±0.76	3.96±0.69	3.70±0.65	**3.11±0.62**
	SGC	20.99±0.02	**3.74±0.29**	4.24±0.14	3.88±0.11
	gfNN	4.38±0.42	4.74±0.67	4.71±1.33	**3.56±1.4**
	APPNP	5.14±0.83	4.47±0.60	4.21±0.75	**3.05±0.55**

Chapter 8

Use case

In this chapter, we apply GNNs calibration to a use case. First we give a short background. Then we show the result and discussion of the application. Finally we set a higher demand for the use case, propose our methods and analyze the results.

8.1 Introduction and motivation

AMEsim [63] is an extensive hydraulic and mechanical simulation platform. It allows engineers to visually model, run and analyze mechanical systems and components.

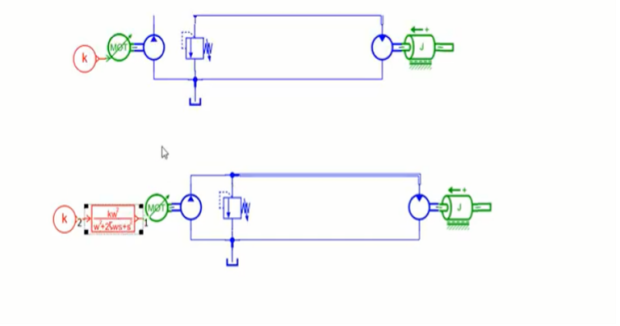


Figure 8.1: A screen shot from AMEsim demo as a simulation example [11]. A pump connects to a motor and a relief valve and so on.

A simple example is shown in Fig. 8.1. AMEsim represents the real devices (Pump, motor and relief valve) as visual devices in the platform.
Our dataset is based on a project which uses GNNs for recommendation systems. In this project, each mechanical device (so called *icon*) is seen

as a node in a graph, with the relationship between two devices as an undirected edge between these two nodes. The dataset is consisted of many small graphs. Each small graph has its own *source node* (also called *source icon*) and *target node* (also called *target icon*). The target node is the node we need to classify with one of 1550 classes, while the source node is the other nodes with known labels connected to the source node in the same small graph. The total train data has about 55k small graphs, with known-label target nodes and source nodes. The total "test" data has about 13k small graphs, with half-half split into validation and test data, either with about 6.5k small graphs.
The task of the project is to a node classification task. Output each-class scores indicate the probability of being that kind of icon. Scores are used to recommend to customers to determine which device is the best option to connect between the target node and the source node according to the ranking among scores.
Our aim is to help provide a reasonable indicator to customers that indicates how confident our model of recommendation system is. Observing calibration of the model can be one option.

8.2 Calibration evaluation

8.2.1 Methods and metrics

Graph models we used include GraphSage [64], GCN [21], and GAT [25].
Calibration methods include Isotonic regression [42], Histogram binning [41], Temperature scaling [16], Bayesian binning into quantiles (BBQ) [32], Meta calibration [65].
Calibration metrics include ECE [16], marginal-ECE [34], Maximum-ECE [16], Brier score [37] and NLL.
Models, methods and metrics are introduced in Chapter 2.

8.2.2 Experiment, results and discussion

Experiment setting. GCN, GAT and GraphSage are used with one layer. Hyperparameters are tuned on each models with learning rate = {0.01,

0.001}, hidden number = {16, 32, 64}, weight decay = 2e-7. Epochs are set to be 60, with early-stopping of patience of 30. Results are averaged with 10-time runs. For calibration methods, ECE is estimated with 15 bins. For meta calibration, miscoverage rate is set to be 0.05. Run time evaluation is performed with one-time run.

Table 8.1: Calibration results for GraphSage. The best result in each metric is displayed in bold.

		Calibration				
Metrics	Uncal	Iso. reg.	His. bin.	Tem. scal.	BBQ	Meta
ECE	12.76	2.31	**1.89**	2.31	2.54	4.26
Max.-ECE	28.40	34.47	**24.70**	34.48	27.26	26.53
Brier scores	0.0640	0.0640	0.0640	0.0640	0.0645	0.0640
NLL	3.77	6.76	3.42	6.76	**3.00**	3.82
Marg.-ECE	0.036	0.014	**0.013**	0.014	0.02	0.048

Table 8.2: Calibration results for GCN. The best result in each metric is displayed in bold.

		Calibration				
Metrics	Uncal	Iso. reg.	His. bin.	Tem. scal.	BBQ	Meta
ECE	7.43	1.94	**1.65**	2.52	2.71	3.87
Max.-ECE	26.63	**24.80**	79.16	42.00	48.80	56.67
Brier scores	0.0570	**0.0536**	0.0585	0.057	0.0572	0.0574
NLL	3.75	6.93	3.51	3.65	**3.09**	3.89
Marg.-ECE	0.034	0.013	**0.011**	0.034	0.0180	0.044

Table 8.3: Calibration results for GAT. The best result in each metric is displayed in bold.

		Calibration				
Metrics	Uncal	Iso. reg.	His. bin.	Tem. scal.	BBQ	Meta
ECE	6.53	**1.79**	2.37	2.22	2.02	4.18
Max.-ECE	48.43	**19.30**	63.00	59.05	85.21	56.72
Brier scores	0.0573	**0.0536**	0.0584	0.0569	0.05760	0.0574
NLL	3.70	8.07	3.89	3.60	3.16	**3.87**
Marg.-ECE	0.033	0.013	**0.011**	0.034	0.018	0.045

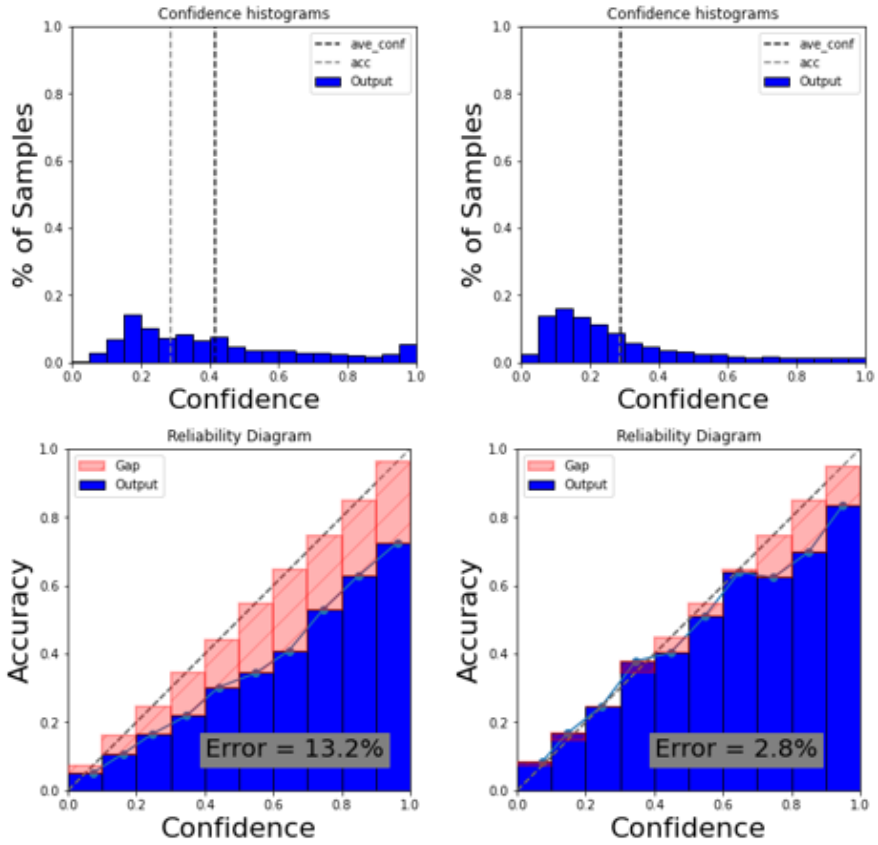


Figure 8.2: An example of confidence histograms and reliability diagram for GraphSage on Amesim. Left top: confidence histograms for uncalibrated scores. Left down: reliability diagram for uncalibrated scores. Right top: confidence histograms after BBQ calibration. Right down: reliability diagram after BBQ calibration.

Results and discussion. The results are shown in Table 8.1, 8.2, 8.3, 8.4 and 8.5. First, GNNs on Amesim dataset exhibit overconfidence, as shown in Fig. 8.2. After calibration, this overconfidence phenomenon gets effectively alleviated. BBQ calibrates GraphSage on Amesim with ECE decreasing from 13.2% to 2.8%. Secondly, comparing among calibration methods, isotonic regression and histogram binning perform the best for most metrics. Thirdly, comparing among models, GraphSage has the best accuracy but the worst uncalibrated performance, while GAT is almost the inverse, as shown in Table. 8.4. This is consistent with observations in Chapter 3 and [66] that attention could help calibrate. After calibration, these three models are calibrated at almost the same level with their best chosen calibration methods. Finally, evaluating the running time of calibration methods, Meta calibration is the fastest method, while BBQ needs the most time. Additionally, according to the result of Chapter 4, we believe that KD could help calibrate due to the overconfidence of Amesim, but not do experiments.

Table 8.4: Model comparison in accuracy, uncalibrated ECE and best-calibrated ECE. The best result in each metric is displayed in bold.

Model	Acc	Uncal. ECE	Best cal. ECE
SAGE	**28.5 ± 0.26**	12.76±0.48	1.89±0.26
GCN	24.35 ± 0.32	7.43±0.47	**1.65±0.37**
GAT	24.42 ± 0.48	**6.53±0.73**	1.79±0.28

Table 8.5: Run time evaluation for calibration

	Calibration				
Metrics	Iso. reg.	His. bin.	Tem. scal.	BBQ	Meta
time/s	41.3	41.1	46.6	660.7	33

8.3 Compatibility evaluation

For a given-class source icon, the connected target icon in the same graph has 1550 possible icons to classify. But not all icons are possible to connect to all icons: due to some physical constraints, only parts of these 1550 target

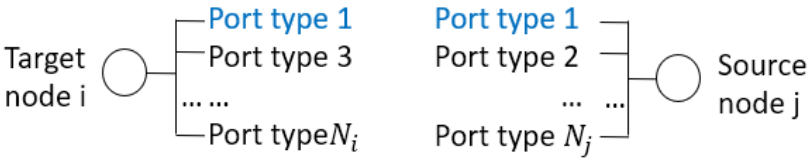


Figure 8.3: An example of target icon i and connected source icon j, with N_i and N_j port types respectively. Only when target and source port types are the same (colored in blue) can two nodes be connected. Port type provides us information from another perspective.

icons can connect to a given source icon. It can be evaluated in terms of *port types*. Each source or target node has its corresponding port types. Fig. 8.3 shows an example of source icon i connected to target icon j. Icon i and j have N_i and N_j port types, repectively. Only when target and source port types are the same (colored in blue) can i and j be connected. These physically possible connected target icons are called *compatible items*, e.g., target icon j is a *compatible item* to source icon i.
Therefore, for each source icon, we have target *ground-truth compatible items*. Compare them to classified rankings in our model, not all the top K (e.g. $K = 10$ or 100) target icons are ground-truth compatible items.
The port type information gives us an opportunity to evaluate probability scores produced by our models using our defined *compatibility metrics* (defined in the next Section).
We further choose a *threshold* for probability scores, with scores above the threshold classified as compatible and below as incompatible. Then these artificially-classified compatible and incompatible items are used to compare to the ground-truth compatible items to evaluate our scores.

8.3.1 Compatibility metrics

Mean ranks of all compatible items is defined as the averaged rank (probability scores ranked in descending) over all compatible items averaged in all test samples:

$$r_{mean} = \frac{1}{MN} \sum_{j}^{M} \sum_{i}^{N} r_{ij}, \tag{8.1}$$

where r_{ij} is the rank for i-th sample and j-th class, N and M being the number of samples and classes.

Averaged compatible number in top K ranks is defined the number of compatible items in top K ranks:

$$\overline{N}_{comp}^{K} = \frac{1}{N}\sum_{i}^{N} n_{comp}^{i,K}, \tag{8.2}$$

where n_{comp}^{i} is the number of compatible items above the K-th rank for sample i.

Precision and Recall [67] are widely-used metrics in statistical learning. Precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned. In our case,

$$\begin{aligned} precision &= \frac{1}{N}\sum_{i}^{N} \frac{n_{comp}^{i,above}}{n^{i,above}} \\ recall &= \frac{1}{N}\sum_{i}^{N} \frac{n_{comp}^{i,above}}{n_{comp}^{i}}, \end{aligned} \tag{8.3}$$

where $n_{comp}^{i,above}$, $n^{i,above}$ and n_{comp}^{i} are respectively the number of compatible items above the threshold, the total number of items above the threshold and the number of compatible items for sample i.

8.3.2 Experiment and results

Setting. Like experiments before, validation and test sets have the same size, each with about 6.5k small graph samples. After a reasonable choice, only 2.8k test samples are left to participate in our metrics [1]. Each experiment is done one time. K in "averaged compatible number in Top K ranks" metric is chosen to be 10 and 100.

[1]Note, the real project has a *filtering step*, to filter out data with the link type of "inConnectedWith" between source and target icons. Therefore, about 4.5k test samples are left. In these samples, there are about 1.8k samples with all target items are compatible, either due to that the source icon's port type is "signal", or all the target icon's port types are "signal". For a effective comparison, *test samples are only chosen in the left about* 2.8*k samples since they are samples that would actually be affected by filtering.* For the calibration part, only methods that can change the order of probability scores are chosen.

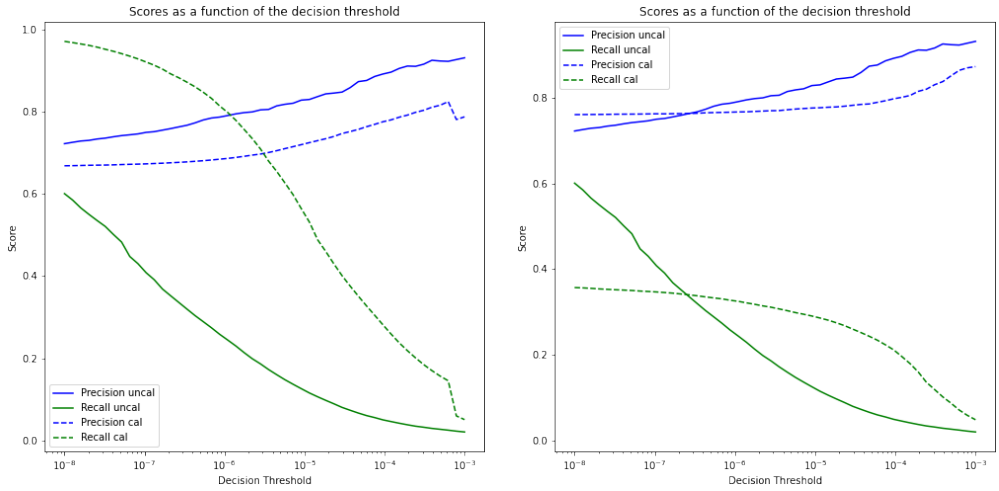


Figure 8.4: Precision and recall before and after calibration. Calibration method is chosen as Meta calibration (Left figure) and BBQ calibration (Right figure) as an example. Solid line represents uncalibrated scores. Dashed line represents calibrated scores.

Table 8.6: Compatibility metrics before calibration and after post-processing calibration. ↓ denotes the lower the better, ↑ denotes the reverse. The best result in each metric is displayed in bold.

	Calibration				
Metrics	Uncal.	Iso. reg.	His. bin.	BBQ	Meta
Comp. ranks↓	774.3	764	**763.2**	765.7	773.1
Comp. # in top 10↑	**9.3**	8.6	8.1	8.1	9.1
Comp. # in top 100↑	**84.4**	74.7	71.9	79.6	84.1

Results and discussions. Table 8.6 shows the results without calibration and after calibration with respect to "compatible ranks" metric and "compatible number in top K ranks" metric. Compared to uncalibrated results, the compatible ranks get lower from 774.3 to around 765 for three calibration methods. This demonstrates calibration can push compatible items to rank forward on the whole. But compatible number in top 10 and 100 ranks get lower. This demonstrates in the sense of top 10 or 100 ranks, the compatible item number decreases.

We further choose a threshold to scores in the range of 10^{-8}-10^{-3}, with scores above the threshold classified as compatible and below as incompatible. Then these artificially-classified compatible and incompatible items are used

to compare to the ground-truth compatible items by using "precision" and "recall". Fig. 8.4 shows precision and recall under thresholds. It shows that as the threshold increases, there are fewer items above the threshold, resulting in lower recall and higher precision. As a result, we can *learn a threshold with respect to the metric we are interested in*. Besides, different thresholds also work in different ranges.

8.4 Multi-class classification vs. binary setting

In multi-class classification setting, scores are input to softmax function, where each-class score interacts with the others. Besides, due to the variety of source nodes (1550 classes), it might be hard to determine a threshold that is suitable for all samples.
Alternatively, in binary setting, scores are input to sigmoid function and then compared with each binary label. This leads to a fair choice of threshold among different classes and samples.
In this section, we explore the multi-class classification setting versus binary setting with respect to accuracy, uncalibrated ECE and compatibility metrics.
Setting. We choose the GraphSage model for this comparison. Parameters are set the same as multi-class classification setting. Experiment is run for one time.

Table 8.7: Multi-class setting vs. binary setting. Metrics used include accuracy, uncal. ECE and three compatibility metrics.

Metrics	Multi-class	Binary
Acc.	0.359	0.358
Uncal. ECE	13.2	9.9
Comp ranks↑	774.3	782.1
Comp # in top 10↑	9.3	9.3
Comp # in top 100↑	84.4	83.9

Results and discussions. Table 8.7 shows the result of two setting. Binary setting reaches almost the same accuracy as the multi-class setting. For compatibility metrics, binary setting also performs very similarly to multi-class setting.

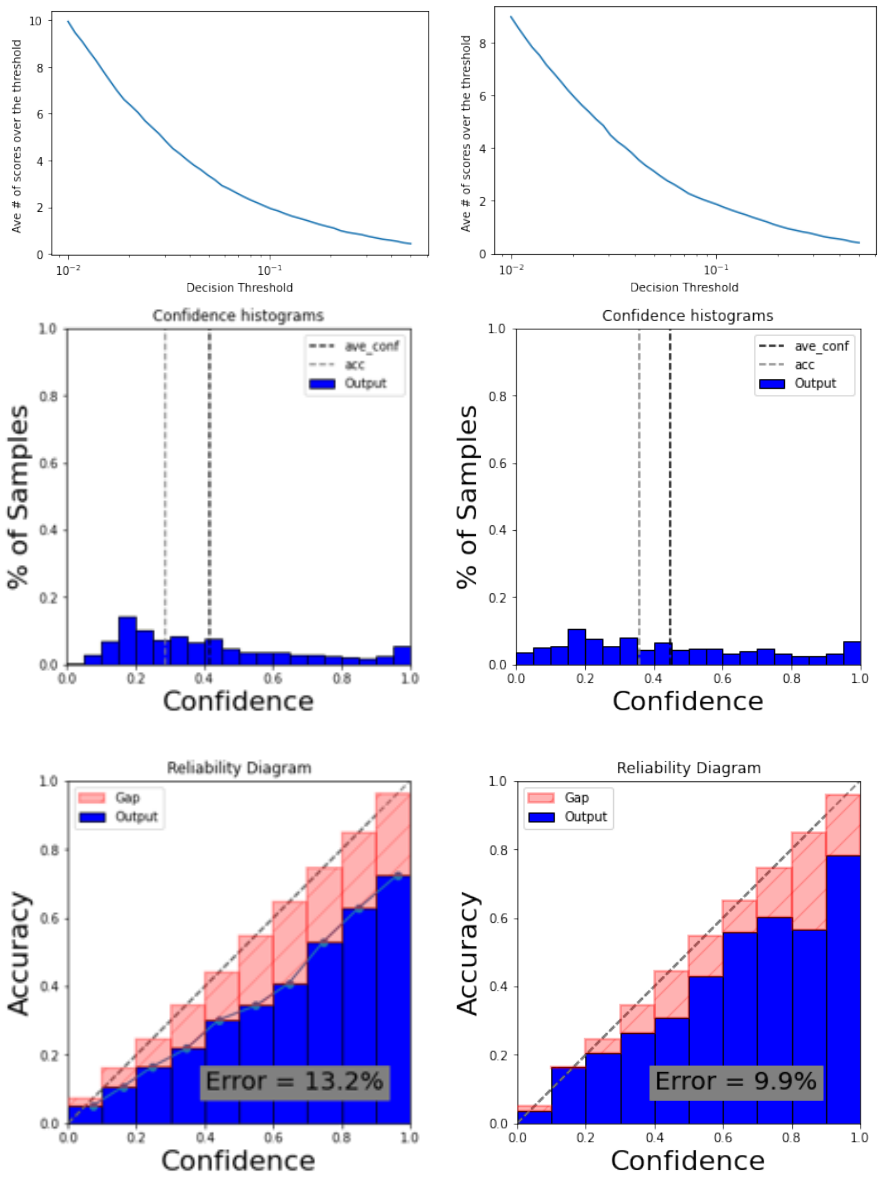


Figure 8.5: An example of all-class scores distribution (top), confidence histograms (middle) and reliability diagram (down) for GraphSage on AMEsim, for multi-class classification setting (left) and binary setting (right).

But when comparing the calibration of the model, binary setting makes model more calibrated, with 13.2% ECE for multi-class setting decreasing to 9.9% ECE for binary setting. Fig. 8.5 shows the diagrams of scores distribution, confidence histograms, and reliability diagrams for these two setting. Like described before, binary setting has an effect of pushing scores down, resulting in the alleviation of overconfidence. *Scores of different class in binary setting do not have as strong interactions as in multi-class setting.* Therefore, binary setting makes model-output probability scores closer to accuracy.

8.5 Conclusion

GNNs on AMEsim exhibit overconfidence. After using existing post processing calibration methods, the overconfidence could be effectively alleviated. Besides, we show that in terms of compatibility metrics, post processing calibration methods are helpful in some metrics but do not fix the problem. Further, the binary setting has almost the same accuracy performance as the multi-class classification setting, but could lead to a more calibrated model.

Chapter 9

Summary and Conclusion

In this thesis, we investigate the model calibration on Graph Neural Networks (GNNs). We focus our tasks on node classification.

First we empirically investigate the calibration performance of benchmark GNNs on benchmark datasets. We show that generally, the calibration performance depends on the dataset and model. When using the hyperparameters from previous publications, GNNs exhibit underconfidence, unlike being overconfident like the other modern neural networks [16]. Existing benchmark post-processing calibration methods could lead to well-calibrated results. Model properties, i.e. depth, width, aggregation and regularization, could lead to different calibration performances respectively. Besides, dataset balancing hardly affects calibration performance. We also contribute to a new-defined metric called balanced-ECE as a proper metric to evaluate under imbalanced classes. (Chapter 3)

Next we evaluate the influence of knowledge distillation (KD) on GNNs to calibration. Yuan et al. proposed to view KD as an adaptive version of label smoothing [19], therefore, should inherit the label smoothing's benefit: improving calibration implicitly [8, 20]. However, our results show that although learning from more informative scores, KD aggravates the underconfidence and helps calibrate overconfident models and miscalibrate underconfident models. Our work is the first paper to state KD does not always help calibrate (Chapter 4).

Then we focus on the influence of calibration from the perspective of graph signal processing. The nature of GNNs, i.e. low-pass filtering, could not increase the hidden calibration information in models. Spectral methods

like GCN outperform MLPs because they have the low-pass filtering ability for both accuracy and calibration (Chapter 5).
Then by adding the calibration loss term to the cross-entropy loss of GNNs, we gain better-calibrated models, with accuracy retained at the same level (Chapter 6).
We propose a novel topology-aware calibration method: ratio-binned scaling, which outperforms most state-of-the-art calibration methods. With a metric of predicted ratio, node-level calibration differences could be distinguished and utilized for further calibration. For future work, it could be interesting to explore and design a new binning scheme for better calibration and exploring the theoretical foundation (Chapter 7).
Finally, we apply our calibration to a real-world use case (AMEsim). We first show that GNNs on Amesim exhibit overconfidence, unlike on citation datasets. After using existing post-processing calibration methods, the overconfidence could be effectively alleviated. With our defined compatibility metrics, post-processing calibration methods are shown to help learn some compatibility information. Further, when comparing the binary and multi-class settings, the binary one could lead to a more calibrated model (Chapter 8).

Appendix A

Unshown figures in Chapter 3

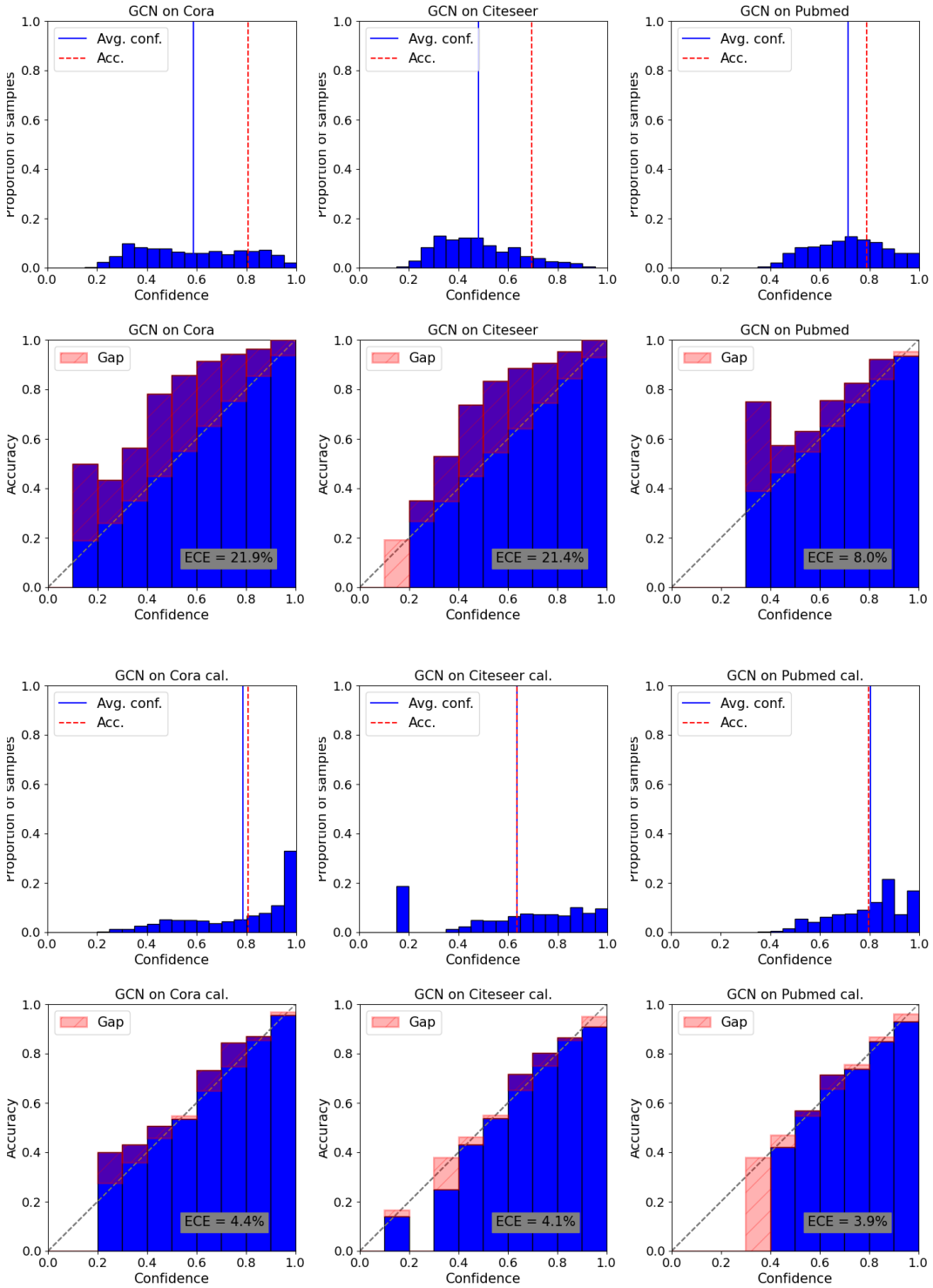


Figure A.1: Histograms and reliability diagrams for GCN. The first and second rows show the result for uncalibrated models. The third and fourth rows show the result for calibrated models. Calibration method is chosen for the best one for the model on the dataset.

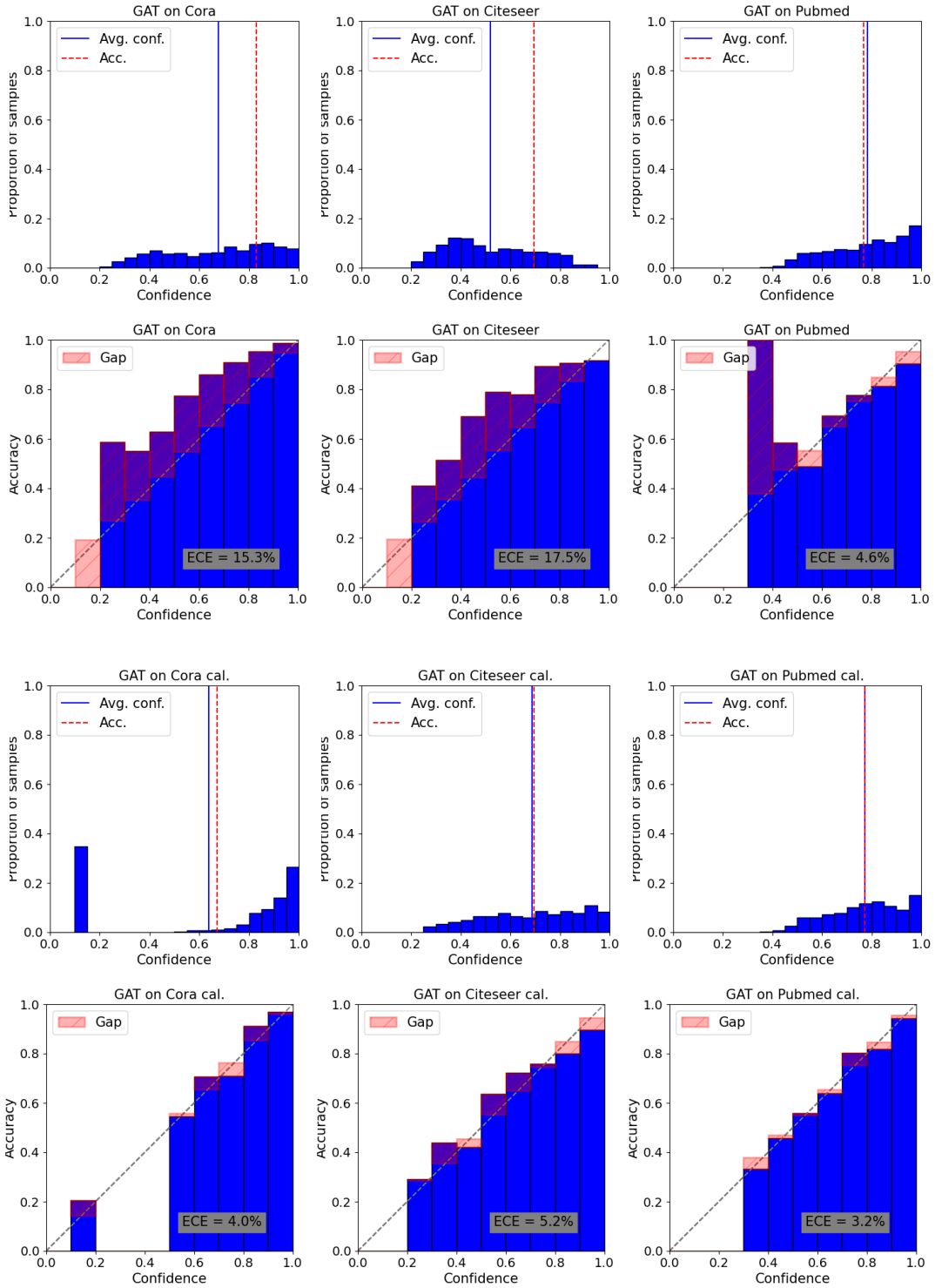


Figure A.2: Histograms and reliability diagrams for GAT. The first and second rows show the result for uncalibrated models. The third and fourth rows show the result for calibrated models. Calibration method is chosen for the best one for the model on the dataset.

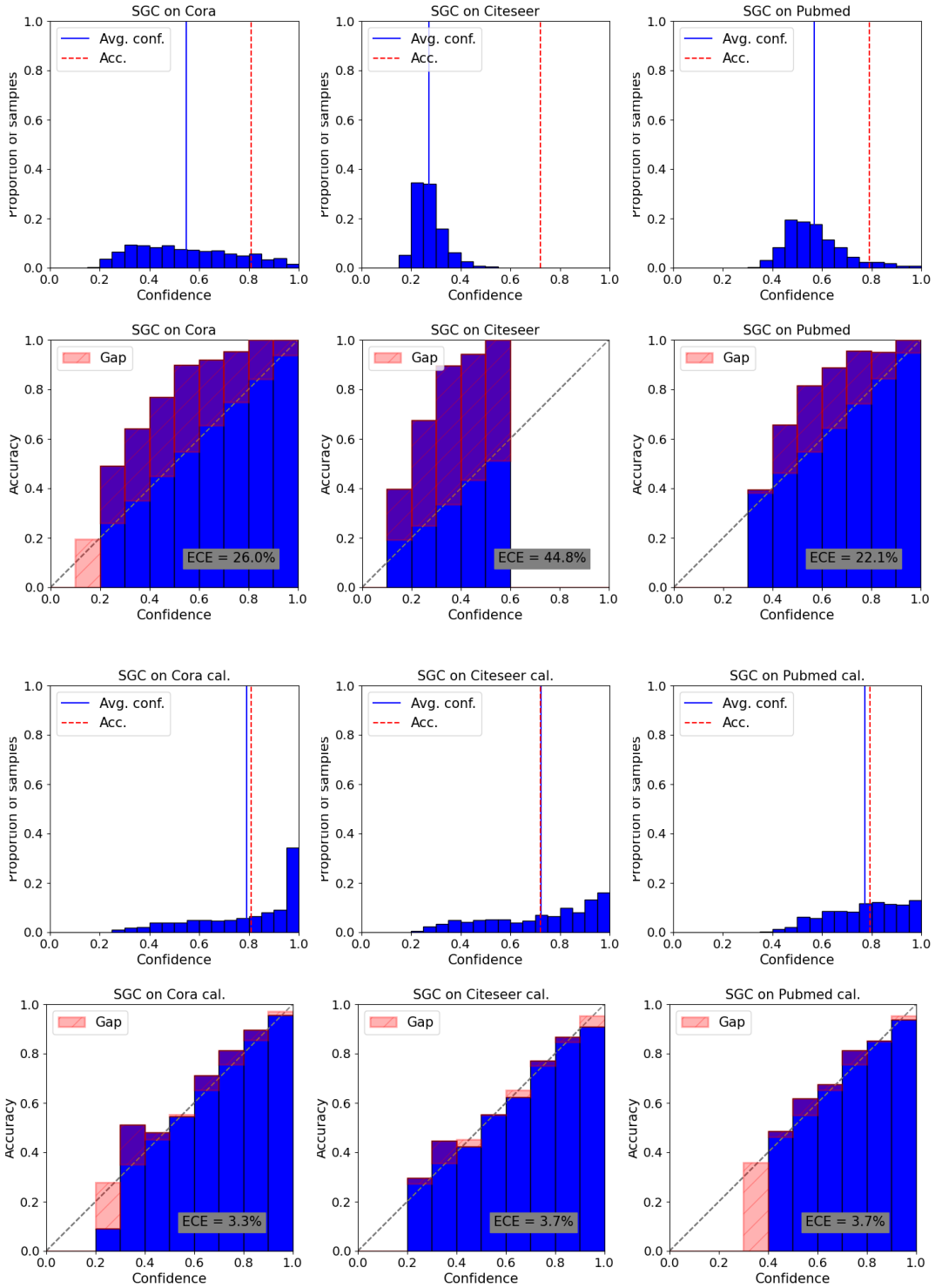


Figure A.3: Histograms and reliability diagrams for SGC. The first and second rows show the result for uncalibrated models. The third and fourth rows show the result for calibrated models. Calibration method is chosen for the best one for the model on the dataset.

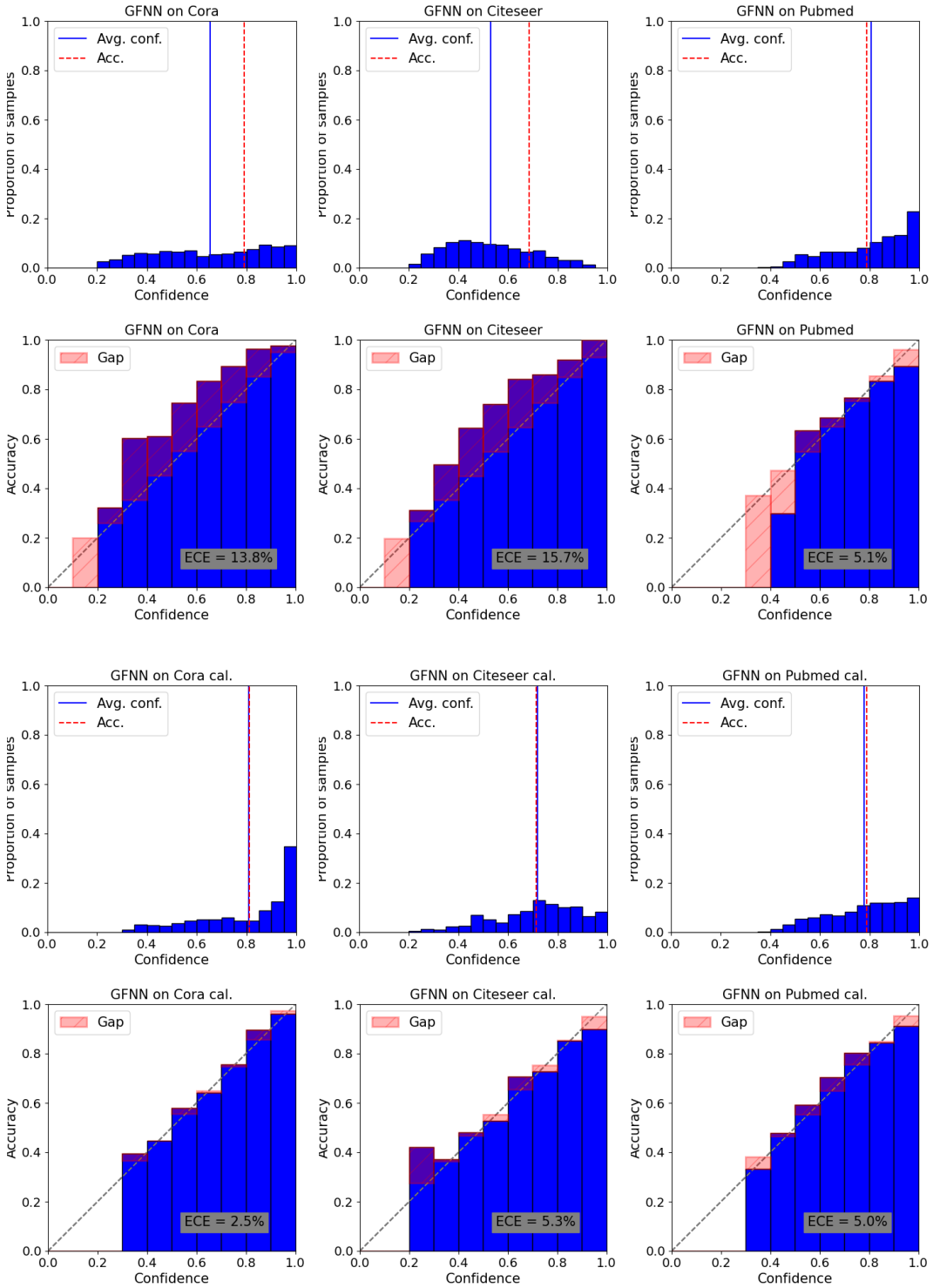


Figure A.4: Histograms and reliability diagrams for gfNN. The first and second rows show the result for uncalibrated models. The third and fourth rows show the result for calibrated models. Calibration method is chosen for the best one for the model on the dataset.

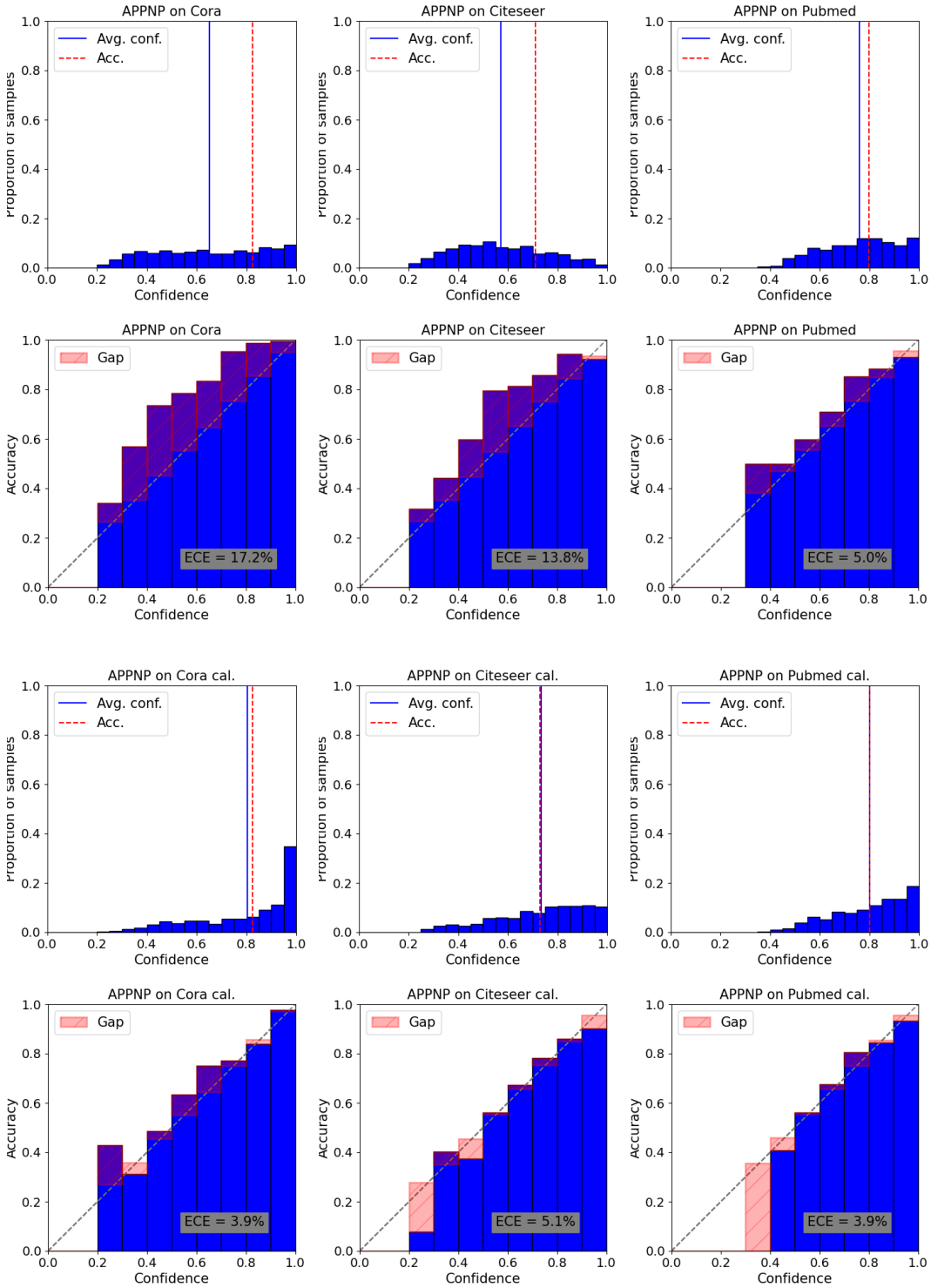


Figure A.5: Histograms and reliability diagrams for APPNP. The first and second rows show the result for uncalibrated models. The third and fourth rows show the result for calibrated models. Calibration method is chosen for the best one for the model on the dataset.

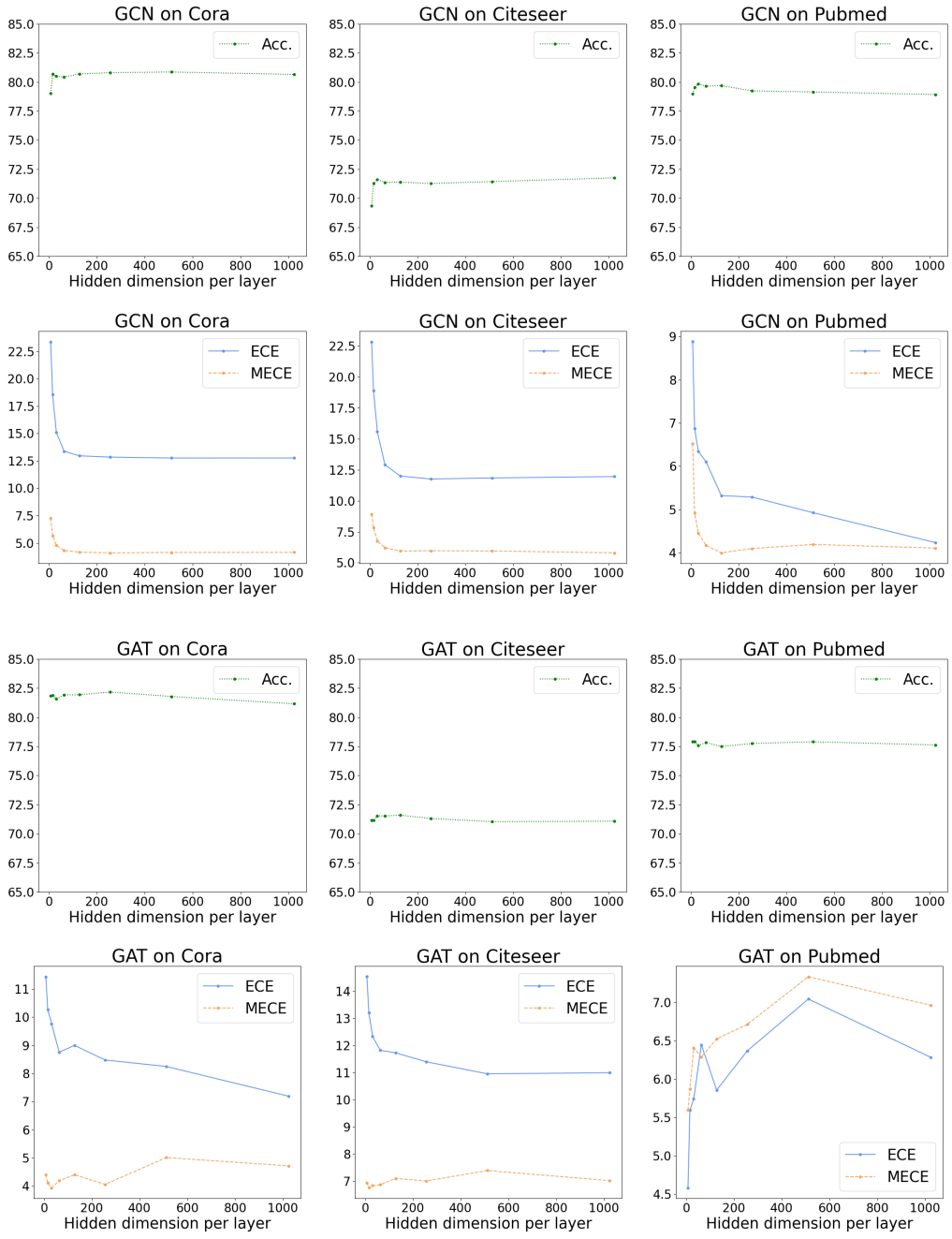


Figure A.6: Influence of width.

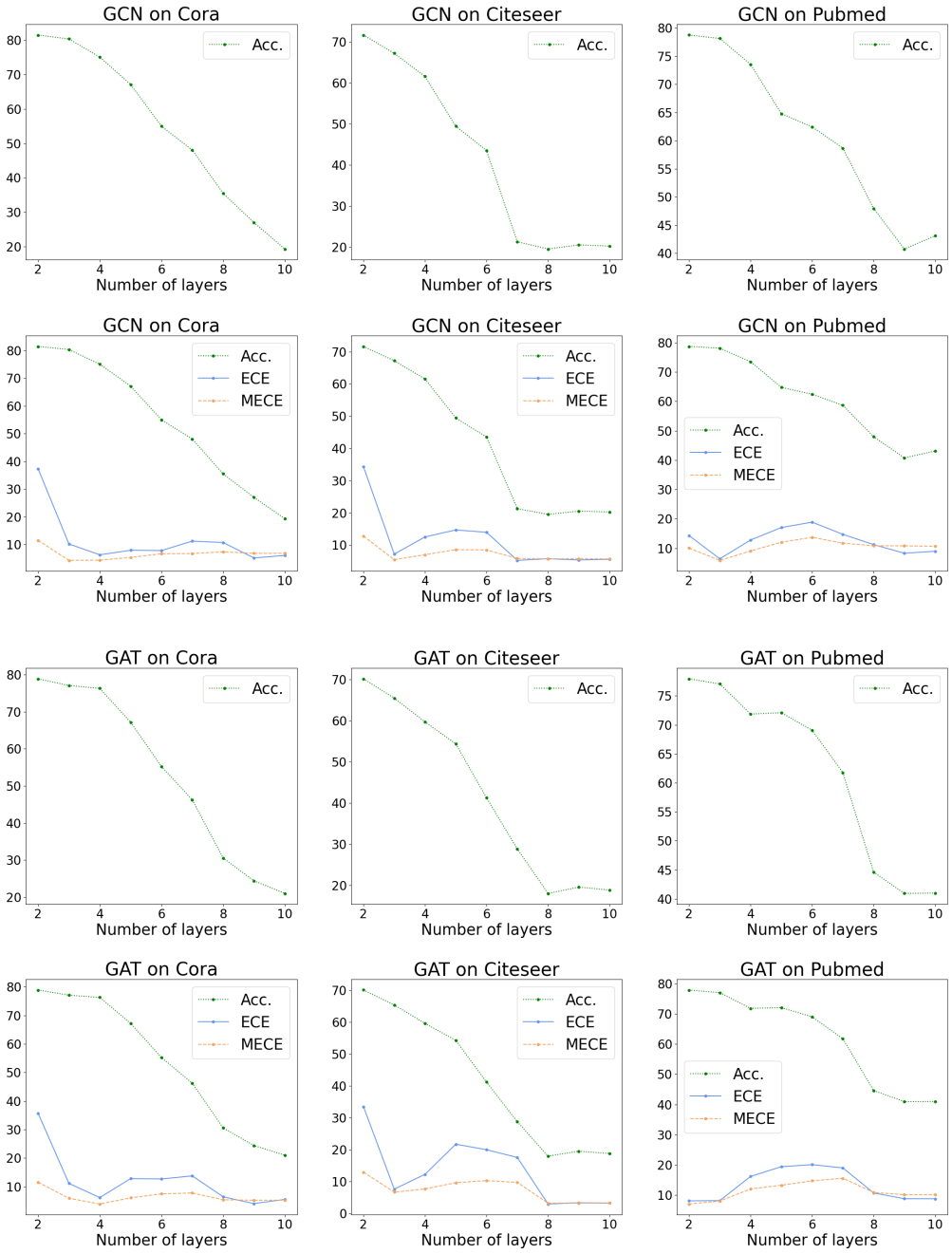


Figure A.7: Influence of depth.

Appendix B

Proof of Theorem 2.2

Theorem 2.2.[1] [38] *Consider joint random variables (X, Z, K), taking values from* $D_X, \mathbb{R}^m$ *and* κ *respectively. Let* $f : D_X \to \Delta^{n-1}$ *be our NN,* $L_1(f, X, K) = -E_{(x,k)\sim(X,K)} log(f_k(x))$ *be the first loss given our model* f *and input* X*,* $L_2(q, Z, K) = -E_{(z,k)\sim(Z,K)} log(q_k(z)) = -E_{(x,k)\sim(X,K)} log(q_k(f(x)))$ *be the second loss given our output* Z*, or equivalently given* f *and* X*.*
A strong version: if

$$f = \underset{f':D_X\to\Delta^{n-1}}{argmin}\ L_1(f', X, K), \tag{B.1}$$

then it is perfect calibrated with respect to f*, i.e.* $P(k|x) = f_k(x)$*.*
A weak version: if

$$\underset{g:\mathbb{R}^m\to\mathbb{R}^m}{argmin}\, L_2(q \circ g, Z, K) = id, \tag{B.2}$$

where id is the identify function, $g : \mathbb{R}^m \to \mathbb{R}^m$ *is a modification, e.g. a post-processing calibration function, then it is perfect calibrated with respect to* g*, i.e.* $P(k|z) = q_k(z)$*.*
An equivalently weak version: if

$$\underset{g:\mathbb{R}^m\to\mathbb{R}^m}{argmin}\, L_2(q \circ g \circ f, X, K) = id, \tag{B.3}$$

then it is perfect calibrated with respect to g*, i.e.* $P(k|z) = q_k(z)$*, or to say,* f *is perfect calibrated.*

[1] Note that the notation here keeps the same in Section 2.2.3 and different from notation in the other parts of thesis.

Proof. This theorem defines the condition of calibration for a model. It gives different conditions to be calibrated for a model: a strong version, a weak version, and a third equivalent version. The strong version is the ideal calibration condition. It is more like a definition. Here we proof the weak version. Then the equivalent version can be simply derived from the weak version.

The weak version states that f is optimal to calibration if replacing f by $g \circ f$, i.e. using a post-processing calibration method g, can not decrease the L_2 loss. Therefore, we want to optimize $L_2(q \circ g, Z, K)$ over g.

The Euler–Lagrange equations in physics states that given the Lagrangian $L = L(t, q, \dot{q})$ of a system, where q is a generalized coordinate, $\dot{q} = \frac{\partial q}{\partial t}$ is a generalized momentum, the following equation holds:

$$\frac{\partial L}{\partial q^i}(t, \boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}^i}(t, \boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) = 0, \quad i = 1, \ldots, n. \tag{B.4}$$

In our case,

$$\begin{aligned} -L_2(q \circ g, Z, K) &= \underset{(z,k)\sim(Z,K)}{E} log(q_k(g(z))) \\ &= \int \sum_{k=1}^{n} p(z,k) \log\left(q_k(g(z))\right) dz \\ &= \int L(z, g, \dot{g}) dz, \end{aligned} \tag{B.5}$$

where $L = L(z, g, \dot{g}) = \sum_{k=1}^{n} p(z,k) \log\left(q_k(g(z))\right)$.

The Euler–Lagrange equations become:

$$\frac{\partial L}{\partial g} = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{g}}. \tag{B.6}$$

$RHS = 0$ since L has no relation to $\dot{g}$. Then since L is optimized when

$g = id$, LHS becomes:

$$\begin{aligned} LHS &= \frac{\partial}{\partial g} \sum_{k=1}^{n} p(z,k) \log \left(q_k(g(z)) \right) \\ &= \sum_{k=1}^{n} p(z,k) \frac{q_k^{'}(\cdot)}{q_k(g(z))} \\ &\overset{g=id}{=} \sum_{k=1}^{n} p(z,k) \frac{q_k^{'}(z)}{q_k(z)} = 0 \end{aligned} \tag{B.7}$$

Since the derivative $q_k^{'}(z)$ is over any class j, we can rewrite it as:

$$\sum_{k=1}^{n} \frac{p(z,k)}{q_k(z)} \frac{\partial q_k}{\partial z_j} = 0 \tag{B.8}$$

Since for $q : \mathbb{R}^m \to \Delta^{n-1}$, we have $\sum_{k=1}^{n} q_k(z) = 1$. Taking partial derivative we have $\sum_{k=1}^{n} \frac{\partial q_k}{\partial z_j} = 0$. Compared with the above equation, we imply:

$$\frac{p(z,k)}{q_k(z)} = constant, \quad for\ any\ k. \tag{B.9}$$

Given $\sum_{k=1}^{n} p(z,k) = c \cdot \sum_{k=1}^{n} q_k(z) = c$, and $\sum_{k=1}^{n} p(z,k) = p(z)$, we have $p(z) = c$. Therefore, $p(k|z) = p(z,k)/p(z) = q_k(z)$. This is the statement of the weak version.

For the third equivalent version, the only difference is that in the loss L_2, the $q \circ g$ becomes $q \circ g \circ f$, and given Z becomes given X. It can be simply derived in the following lemma.

Lemma B.1. [38] Consider joint random variables (X, Z), taking values from D_X and $\mathbb{R}^m$ respectively. Let $f : D_X \to D_Z$ be our model, satisfying $Z = f(X)$. Let $L = L(f(x))$ be any function. We have:

$$\underset{x \sim X}{E} L(f(x)) = \underset{z \sim Z}{E} L(z). \tag{B.10}$$

Bibliography

[1] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artifical Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[2] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[3] Y. Liu, M. Hildebrandt, M. Joblin, M. Ringsquandl, R. Raissouni, and V. Tresp, "Neural multi-hop reasoning with logical rules on biomedical knowledge graphs," in *European Semantic Web Conference*. Springer, 2021, pp. 375–391.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[5] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[6] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[8] R. Müller, S. Kornblith, and G. Hinton, "When does label smoothing help?" *arXiv preprint arXiv:1906.02629*, 2019.

[9] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[10] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*. PMLR, 2019, pp. 6861–6871.

[11] Simcenter amesim demo. [Online]. Available: https://www.youtube.com/watch?v=4ja0HkBTxUc&t=94s

[12] A. D. Becke, "Perspective: Fifty years of density-functional theory in chemical physics," *The Journal of chemical physics*, vol. 140, no. 18, p. 18A301, 2014.

[13] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Scientific data*, vol. 1, no. 1, pp. 1–7, 2014.

[14] Z. Savitsky. The next big thing: the use of graph neural networks to discover particles. [Online]. Available: https://news.fnal.gov/2020/09/the-next-big-thing-the-use-of-graph-neural-networks-to-discover-particles/

[15] J. Shlomi, P. Battaglia, and J.-R. Vlimant, "Graph neural networks in particle physics," *Machine Learning: Science and Technology*, vol. 2, no. 2, p. 021001, 2020.

[16] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1321–1330.

[17] P. Tabacof and L. Costabello, "Probability calibration for knowledge graph embedding models," *arXiv preprint arXiv:1912.10000*, 2019.

[18] L. Teixeira, B. Jalaian, and B. Ribeiro, "Are graph neural networks miscalibrated?" *arXiv preprint arXiv:1905.02296*, 2019.

[19] L. Yuan, F. E. Tay, G. Li, T. Wang, and J. Feng, "Revisiting knowledge distillation via label smoothing regularization," in *Proceedings of the*

IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 3903–3911.

[20] J. Tang, R. Shivanna, Z. Zhao, D. Lin, A. Singh, E. H. Chi, and S. Jain, "Understanding and improving knowledge distillation," *arXiv preprint arXiv:2002.03532*, 2020.

[21] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[22] H. Nt and T. Maehara, "Revisiting graph neural networks: All we have is low-pass filters," *arXiv preprint arXiv:1905.09550*, 2019.

[23] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[24] P. D. Hoff, A. E. Raftery, and M. S. Handcock, "Latent space approaches to social network analysis," *Journal of the american Statistical association*, vol. 97, no. 460, pp. 1090–1098, 2002.

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[26] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5453–5462.

[27] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.

[28] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[29] J. Yang, B. Ribeiro, and J. Neville, "Stochastic gradient descent for relational logistic regression via partial network crawls," *arXiv preprint arXiv:1707.07716*, 2017.

[30] M. H. DeGroot and S. E. Fienberg, "The comparison and evaluation of forecasters," *Journal of the Royal Statistical Society: Series D (The Statistician)*, vol. 32, no. 1-2, pp. 12–22, 1983.

[31] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 625–632.

[32] M. P. Naeini, G. Cooper, and M. Hauskrecht, "Obtaining well calibrated probabilities using bayesian binning," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[33] K. Gupta, A. Rahimi, T. Ajanthan, T. Mensink, C. Sminchisescu, and R. Hartley, "Calibration of neural networks using splines," *arXiv preprint arXiv:2006.12800*, 2020.

[34] A. Kumar, P. Liang, and T. Ma, "Verified uncertainty calibration," *arXiv preprint arXiv:1909.10155*, 2019.

[35] A. Rahimi, A. Shaban, C.-A. Cheng, R. Hartley, and B. Boots, "Intra order-preserving functions for calibration of multi-class neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 456–13 467, 2020.

[36] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[37] G. W. Brier *et al.*, "Verification of forecasts expressed in terms of probability," *Monthly weather review*, vol. 78, no. 1, pp. 1–3, 1950.

[38] A. Rahimi, K. Gupta, T. Ajanthan, T. Mensink, C. Sminchisescu, and R. Hartley, "Post-hoc calibration of neural networks," *arXiv preprint arXiv:2006.12807*, 2020.

[39] Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Bayesian_inference#:~:text=Bayesian%20inference%20is%20a%20method,and%20especially%20in%20mathematical%20statistics./

[40] J. Platt *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[41] B. Zadrozny and C. Elkan, "Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers," in *Icml*, vol. 1. Citeseer, 2001, pp. 609–616.

[42] ——, "Transforming classifier scores into accurate multiclass probability estimates," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 694–699.

[43] C. Xing, S. Arik, Z. Zhang, and T. Pfister, "Distance-based learning from errors for confidence calibration," *arXiv preprint arXiv:1912.01730*, 2019.

[44] J. Mukhoti, V. Kulharia, A. Sanyal, S. Golodetz, P. H. Torr, and P. K. Dokania, "Calibrating deep neural networks using focal loss," *arXiv preprint arXiv:2002.09437*, 2020.

[45] S. Thulasidasan, G. Chennupati, J. Bilmes, T. Bhattacharya, and S. Michalak, "On mixup training: Improved calibration and predictive uncertainty for deep neural networks," *arXiv preprint arXiv:1905.11001*, 2019.

[46] C. Tomani and F. Buettner, "Towards trustworthy predictions from deep neural networks with fast adversarial calibration," *arXiv preprint arXiv:2012.10923*, 2020.

[47] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[48] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of*

the IEEE conference on computer vision and pattern recognition, 2016, pp. 2818–2826.

[49] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[50] J. Klicpera, J. Groß, and S. Günnemann, "Directional message passing for molecular graphs," *arXiv preprint arXiv:2003.03123*, 2020.

[51] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[52] J. Bergstra, D. Yamins, D. D. Cox *et al.*, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in science conference*, vol. 13. Citeseer, 2013, p. 20.

[53] M. Kull, M. Perello-Nieto, M. Kängsepp, H. Song, P. Flach *et al.*, "Beyond temperature scaling: Obtaining well-calibrated multiclass probabilities with dirichlet calibration," *arXiv preprint arXiv:1910.12656*, 2019.

[54] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[55] F. Harrell. Classification vs. prediction. [Online]. Available: https://www.fharrell.com/post/classification/

[56] C. Yang, J. Liu, and C. Shi, "Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework," in *Proceedings of the Web Conference 2021*, 2021, pp. 1227–1237.

[57] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," *arXiv preprint arXiv:2002.05287*, 2020.

[58] Z. Xiaojin and G. Zoubin, "Learning from labeled and unlabeled data with label propagation," *Tech. Rep., Technical Report CMU-CALD-02–107, Carnegie Mellon University*, 2002.

[59] H. Li, "Exploring knowledge distillation of deep neural."

[60] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[61] H. Zhu and P. Koniusz, "Simple spectral graph convolution," in *International Conference on Learning Representations*, 2020.

[62] D. Bo, X. Wang, C. Shi, and H. Shen, "Beyond low-frequency information in graph convolutional networks," *arXiv preprint arXiv:2101.00797*, 2021.

[63] Simcenter amesim. [Online]. Available: https://www.plm.automation.siemens.com/global/en/products/simcenter/simcenter-amesim.html

[64] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[65] X. Ma and M. B. Blaschko, "Meta-cal: Well-controlled post-hoc calibration by ranking," *arXiv preprint arXiv:2105.04290*, 2021.

[66] S. Desai and G. Durrett, "Calibration of pre-trained transformers," *arXiv preprint arXiv:2003.07892*, 2020.

[67] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 233–240.