JuryGCN: Quantifying Jackknife Uncertainty on Graph Convolutional Networks

Jian Kang*
jiank2@illinois.edu
University of Illinois at Urbana-Champaign

Qinghai Zhou*
qinghai2@illinois.edu
University of Illinois at Urbana-Champaign

Hanghang Tong
htong@illinois.edu
University of Illinois at Urbana-Champaign

ABSTRACT

Graph Convolutional Network (GCN) has exhibited strong empirical performance in many real-world applications. The vast majority of existing works on GCN primarily focus on the accuracy while ignoring how confident or uncertain a GCN is with respect to its predictions. Despite being a cornerstone of trustworthy graph mining, uncertainty quantification on GCN has not been well studied and the scarce existing efforts either fail to provide deterministic quantification or have to change the training procedure of GCN by introducing additional parameters or architectures. In this paper, we propose the first frequentist-based approach named JuryGCN in quantifying the uncertainty of GCN, where the key idea is to quantify the uncertainty of a node as the width of confidence interval by a jackknife estimator. Moreover, we leverage the influence functions to estimate the change in GCN parameters without re-training to scale up the computation. The proposed JuryGCN is capable of quantifying uncertainty deterministically without modifying the GCN architecture or introducing additional parameters. We perform extensive experimental evaluation on real-world datasets in the tasks of both active learning and semi-supervised node classification, which demonstrate the efficacy of the proposed method.

CCS CONCEPTS

• **Information systems → Data mining**.

KEYWORDS

Graph neural networks, uncertainty quantification, jackknife

ACM Reference Format:
Jian Kang, Qinghai Zhou, and Hanghang Tong. 2022. JuryGCN: Quantifying Jackknife Uncertainty on Graph Convolutional Networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539286

1 INTRODUCTION

Graph Convolutional Network (GCN) has become a prevalent learning paradigm in many real-world applications, including financial fraud detection [45], drug discovery [19] and traffic prediction [7]. To date, the vast majority of existing works do not take into account the uncertainty of a GCN regarding its prediction, which is alarming especially in high-stake scenarios. For example, in automated financial fraud detection, it is vital to let expert banker to take controls if a GCN-based detector is highly uncertain about its predictions in order to prevent wrong decisions on suspending banking account(s).

A well established study on uncertainty quantification of GCN could bring several crucial benefits. First, it is a cornerstone in trustworthy graph mining. Uncertainty quantification aims to understand to what extent the model is likely to be incorrect, and thus provides natural remedy to questions like *how uncertain is a GCN in its own predictions?* Second, an accurate quantification of GCN uncertainty could potentially answer *how to improve GCN predictions by leveraging its uncertainty* in many graph mining tasks. For example, in active learning on graphs, nodes with high uncertainty could be selected as the most valuable node to query the oracle; in node classification, the node uncertainty could help calibrate the confidence of GCN predictions, thereby improving the overall classification accuracy.

Important as it could be, very few studies on uncertainty quantification of GCN exist, which mainly focuses on two different directions: Bayesian-based approaches and deterministic quantification-based approaches. Regarding Bayesian-based approaches [21, 48], they either drops edge(s) with certain sampling strategies or leverages random graph model (e.g., stochastic block model) to assign edge probabilities for training. However, these models fall short in explicitly quantifying the uncertainty on model predictions. Another type of methods, i.e., deterministic quantification-based approaches [30, 40, 49], directly quantifies uncertainty by parameterizing a Dirichlet distribution as prior to estimate the posterior distribution under a Bayesian framework. Nevertheless, it changes the training procedures of a graph neural network by introducing additional parameters (e.g., parameters for Dirichlet distribution) or additional architectures (e.g., teacher network) in order to precisely estimate the uncertainty.

To address the aforementioned limitations, we provide the first study on frequentist-based analysis of the GCN uncertainty, which we term as the JuryGCN problem. Building upon the general principle of jackknife (leave-one-out) resampling [34], the jackknife uncertainty of a node is defined as the width of confidence interval constructed by a jackknife estimator when leaving the corresponding node out. In order to estimate the GCN parameters without exhaustively re-training GCN, we leverage influence functions [28] to quantify the change in GCN parameters by infinitesimally upweighting the loss of a training node. Compared with existing works, our method brings several advantages. First, our method provides deterministic uncertainty quantification, which

*Both authors contributed equally to this research.

is not available in Bayesian-based approaches [21, 48]. Second, different from existing works on deterministic uncertainty quantification [30, 40, 49], our method does not introduce any additional parameters or components in the GCN architecture. Third, our method can provide *post-hoc* uncertainty quantification. As long as the input graph and a GCN are provided, our method can *always* quantify node uncertainty without any epoch(s) of model training.

The major contributions of this paper are summarized as follows.

- **Problem definition.** To our best knowledge, we provide the first frequentist-based analysis of GCN uncertainty and formally define the JuryGCN problem.
- **Algorithm and analysis.** We propose JuryGCN to quantify jackknife uncertainty on GCN. The key idea is to leverage a jackknife estimator to construct a leave-one-out predictive confidence interval for each node, where the leave-one-out predictions are estimated using the influence functions with respect to model parameters.
- **Experimental evaluations.** We demonstrate the effectiveness of JuryGCN through extensive experiments on real-world graphs in active learning on node classification and semi-supervised node classification.

2 PROBLEM DEFINITION

In this section, we first introduce preliminary knowledge on the Graph Convolutional Network (GCN), predictive uncertainty and jackknife resampling. Then, we formally define the problem of jackknife uncertainty quantification on GCN (JuryGCN).

Unless otherwise specified, we use bold upper-case letters for matrices (e.g., $\mathbf{A}$), bold lower-case letters for vectors (e.g., $\mathbf{x}$), calligraphic letters for sets (e.g., $\mathcal{G}$) and fraktur font for high-dimensional tensors ($\mathfrak{H}$). We use supercript ^{T} for matrix transpose and superscript $^{-1}$ for matrix inversion, i.e., $\mathbf{A}^T$ and $\mathbf{A}^{-1}$ are the transpose and inverse of $\mathbf{A}$, respectively. We use conventions similar to PyTorch in Python for indexing. For example, $\mathbf{A}[i,j]$ represents the entry of $\mathbf{A}$ at the i-th row and j-th column; $\mathbf{A}[i,:]$ and $\mathbf{A}[:,j]$ demonstrate the i-th row and j-th column of $\mathbf{A}$, respectively.

2.1 Preliminaries

1 – Graph Convolutional Network (GCN) Let $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$ denote a graph whose node set is $\mathcal{V}$, adjacency matrix is $\mathbf{A}$ and node feature matrix is $\mathbf{X}$. For the l-th hidden layer in an L-layer GCN, we assume $\mathbf{E}^{(l)}$ is the output node embeddings (where $\mathbf{E}^{(0)} = \mathbf{X}$) and $\mathbf{W}^{(l)}$ is the weight matrix. Mathematically, the graph convolution at the l-th hidden layer can be represented by $\mathbf{E}^{(l)} = \sigma(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)})$ where σ is the activation and $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is the renormalized graph Laplacian with $\tilde{\mathbf{D}}$ being the degree matrix of $(\mathbf{A}+\mathbf{I})$.

2 – Uncertainty quantification is one of the cornerstones in safe-critical applications. It provides accurate quantification on how confident a mining model is towards its predictions. In general, uncertainty can be divided into two types: *aleatoric* uncertainty and *epistemic* uncertainty [1]. Aleatoric uncertainty (or data uncertainty) refers to the variability in mining results due to the inherent randomness in input data, which is irreducible due to complexity of input data (e.g., noise); whereas epistemic uncertainty (or model uncertainty) measures how well the mining model fits the training data due to the lack of knowledge on the optimal model parameters, which is reducible by increasing the size of training data.

3 – Jackknife resampling is a classic method to estimate the bias and variance of a population [41]. It often relies on a jackknife estimator which is built by leaving out an observation from the entire population (i.e., leave-one-out) and evaluating the error of the model re-trained on the held-out population. Suppose we have (1) a set of n data points $\mathcal{D} = \{(\mathbf{x}_i, y_i)|i = 1,\dots,n\}$, (2) a test point $(\mathbf{x}_{\text{test}}, y_{\text{test}})$, (3) a mining model $f_\theta()$ parameterized by θ (e.g., a neural network) where $f_\theta(\mathbf{x})$ is the prediction of input feature $\mathbf{x}$ and (4) a target coverage level $(1-\alpha)$ such that the label y is covered by the predictive confidence interval with probability $(1-\alpha)$. Mathematically, the confidence interval constructed by the naive jackknife [13] is upper bounded by $\mathbb{C}^+(\mathbf{x}_{\text{test}}) = Q_{1-\alpha}(\mathcal{R}^+)$ and lower bounded by $\mathbb{C}^-(\mathbf{x}_{\text{test}}) = Q_\alpha(\mathcal{R}^-)$, where Q_α finds the α quantile of a set and $\mathcal{R}^\gamma = \{f_\theta(\mathbf{x}_{\text{test}}) + \gamma \cdot |y_{\text{test}} - f_{\theta_{-i}}(\mathbf{x}_{\text{test}})||i = 1,\dots,n\}$ for $\gamma \in \{-,+\}$ and $|y_{\text{test}} - f_{\theta_{-i}}(\mathbf{x}_{\text{test}})|$ is the error residual of the re-trained model on the dataset $\mathcal{D}\backslash\{(\mathbf{x}_i, y_i)\}$ (i.e., parameterized by θ_{-i}).[1] Hence, $\mathcal{R}^+$ and $\mathcal{R}^-$ represent the sets of upper and lower uncertainty bound on the original model prediction (i.e., $f_\theta(\mathbf{x}_{\text{test}})$). Furthermore, jackknife+ [3] constructs the predictive confidence interval for exchangeable data as

$$\mathbb{C}^+(\mathbf{x}_{\text{test}}) = Q_{1-\alpha}(\mathcal{P}^+) \qquad \mathbb{C}^-(\mathbf{x}_{\text{test}}) = Q_\alpha(\mathcal{P}^-) \tag{1}$$

where $\mathcal{P}^\gamma$ for $\gamma \in \{-,+\}$ is defined as $\mathcal{P}^\gamma = \{f_{\theta_{-i}}(\mathbf{x}_{\text{test}}) + \gamma \cdot |y_i - f_{\theta_{-i}}(\mathbf{x}_i)||i = 1,\dots,n\}$. Similarly, $\mathcal{P}^-$ and $\mathcal{P}^+$ represent the sets of the lower and upper uncertainty bound of the leave-one-out prediction $f_{\theta_{-i}}(\mathbf{x}_{\text{test}})$, respectively. With the assumption on data exchangeability, it yields a $(1-2\alpha)$ coverage rate theoretically.

2.2 Problem Definition

Existing works on deterministic uncertainty quantification on a graph neural network (GNN) mainly rely on changing the training procedures of a vanilla GNN (i.e., graph neural network without consideration of uncertainty) [40, 49]. As such, given a well-trained GNN, it requires epoch(s) of re-training to quantify its uncertainty. Nevertheless, it would cost a lot of computational resources to re-train it, especially when the model has already been deployed in an operational environment. Additionally, it remains a challenging problem to further comprehend the predictive results of GNN from the perspective of uncertainty, and to answer the following question: *to what extent the GNN model is confident of the current prediction?* Therefore, it is essential to investigate the uncertainty quantification in a post-hoc manner, i.e., quantifying uncertainty without further (re-)training on the model.

Regarding post-hoc uncertainty quantification for IID (i.e., non-graph) data, Alaa and van der Schaar [2] propose a frequentist-based method inspired by jackknife resampling. It uses high-order influence functions to quantify the impact of a data point on the underlying neural network. Given that the parameters of a neural network are derived by learning with data points, high-order influence functions are capable of understanding how much a data point will affect the model parameters. Then, the change in model parameters can be used to infer the uncertainty of the corresponding data point on the neural network by a jackknife estimator [3]. Under mild assumption (such as the algorithmic stability assumption and the IID/exchangebility of data), the naive jackknife estimator [13] and its variants [3] bear strong theoretical guarantee in terms of

[1]We use γ to represent the symbol before the leave-one-out error, i.e., $f_\theta(\mathbf{x}_{\text{test}}) - |y_{\text{test}} - f_{\theta_{-i}}(\mathbf{x}_{\text{test}})|$ when $\gamma = -$, or $f_\theta(\mathbf{x}_{\text{test}}) + |y_{\text{test}} - f_{\theta_{-i}}(\mathbf{x}_{\text{test}})|$ otherwise.

the coverage such that the confidence interval will cover the true model parameters with a high probability.

Building upon the jackknife resampling [34] and the general principle outlined in [2], we seek to bridge the gap between frequentist-based uncertainty quantification and graph neural networks. To be specific, given an input graph and a GCN, we aim to estimate the uncertainty of a node as the impact of leaving out its loss when computing the overall loss. Formally, we define the problem of jackknife uncertainty quantification on GCN, which is referred to as JuryGCN problem.

Problem 1. *JuryGCN: Jackknife Uncertainty Quantification on Graph Convolutional Network*

Given: (1) An undirected graph $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$; (2) an L-layer GCN with the set of weights Θ; (3) a task-specific loss function $R(\mathcal{G}, \mathcal{Y}, \Theta)$ where $\mathcal{Y}$ is the set of node labels.

Find: An uncertainty score $\mathbb{U}_\Theta(u)$ for any node u in graph $\mathcal{G}$ with respect to the GCN parameters Θ and the task-specific loss function $R(\mathcal{G}, \mathcal{Y}, \Theta)$.

3 JURYGCN: MEASURE AND COMPUTATION

In this section, we start by discussing the general strategy of quantifying jackknife uncertainty with influence functions, and formally define the node jackknife uncertainty. After that, we present mathematical analysis on the influence function computation for GCN.

3.1 Jackknife Uncertainty of GCN

In this paper, we consider an L-layer GCN with ReLU activation for node-level tasks (e.g., node classification). We further assume that the nodes of the input graph $\mathcal{G}$ are exchangeable data.[2]

We first observe that the loss function of many node-level tasks can often be decomposed into a set of node-specific subproblems. Mathematically, it can be written as

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}}\, R(\mathcal{G}, \mathcal{Y}_{\text{train}}, \Theta) = \underset{\Theta}{\operatorname{argmin}}\, \frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{v \in \mathcal{V}_{\text{train}}} r(v, \mathbf{y}_v, \Theta) \tag{2}$$

where $\mathcal{V}_{\text{train}} \subseteq \mathcal{V}$ is the set of training nodes, $|\mathcal{V}_{\text{train}}|$ is the number of training nodes, $\mathcal{Y}_{\text{train}} = \{\mathbf{y}_v | v \in \mathcal{V}_{\text{train}}\}$ is the set of ground-truth training labels and $\mathbf{y}_v$ is the label of node v. In this case, the overall loss function $R(\mathcal{G}, \mathcal{Y}_{\text{train}}, \Theta)$ is decomposed into several subproblems, each of which minimizes the node-specific loss function $r(v, \mathbf{y}_v, \Theta)$ for a node v. An example is the cross entropy with node-specific loss as $r(v, \mathbf{y}_v, \Theta) = -\sum_{i=1}^{c} \mathbf{y}_v[c] \log\left(GCN(v, \Theta)[c]\right)$, where c is the number of classes, $GCN(v, \Theta)$ is the vector of predicted probabilities for each class using the GCN with parameters Θ. If we upweight the importance of optimizing the loss of a node i with some small constant ϵ, we have the following loss function.

$$\Theta^*_{\epsilon,i} = \underset{\Theta}{\operatorname{argmin}}\, \epsilon r(i, \mathbf{y}_i, \Theta) + \frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{v \in \mathcal{V}_{\text{train}}} r(v, \mathbf{y}_v, \Theta) \tag{3}$$

Influence function is a powerful approach to evaluate the dependence of the estimator on the value of the data examples [28, 52]. In order to obtain $\Theta^*_{\epsilon,i}$ without re-training the GCN, we leverage the influence function [28], which is essentially the Taylor expansion over the model parameters.

$$\Theta^*_{\epsilon,i} \approx \Theta^* + \epsilon \mathbb{I}_{\Theta^*}(i) \tag{4}$$

where $\mathbb{I}_{\Theta^*}(i) = \frac{d\Theta^*_{\epsilon,i}}{d\epsilon}|_{\epsilon=0}$ is the influence function with respect to node i. The influence function $\mathbb{I}_{\Theta^*}(i)$ can be further computed using the classical result in [9] as

$$\mathbb{I}_{\Theta^*}(i) = \mathbf{H}^{-1}_{\Theta^*} \nabla_\Theta r(i, \mathbf{y}_i, \Theta^*) \tag{5}$$

where $\mathbf{H}_{\Theta^*} = \frac{1}{|\mathcal{V}_{\text{train}}|} \nabla^2_\Theta R(\mathcal{G}, \mathcal{Y}_{\text{train}}, \Theta^*)$ is the Hessian matrix with respect to model parameters Θ^*. For a training node i, by setting $\epsilon = -\frac{1}{|\mathcal{V}_{\text{train}}|}$, Eq. (4) efficiently estimates the leave-one-out (LOO) parameters $\Theta^*_{\epsilon,i}$ if leaving out the loss of node i. After that, by simply switching the original model parameters to $\Theta^*_{\epsilon,i}$, we estimate the leave-one-out error err_i of node i as follows.

$$err_i = \|\mathbf{y}_i - GCN(i, \Theta^*_{\epsilon,i})\|_2 \tag{6}$$

where $GCN(u, \Theta^*_{\epsilon,i})$ represents the output of node u using the GCN with leave-one-out parameters $\Theta^*_{\epsilon,i}$.

With Eq. (6), we use jackknife+ [3], which requires the data to be exchangeable instead of IID, to construct the confidence interval of node u. Mathematically, the lower bound $\mathbb{C}^-_\Theta(u)$ and upper bound $\mathbb{C}^+_\Theta(u)$ of the predictive confidence interval of node u are

$$\begin{aligned} \mathbb{C}^-_{\Theta^*}(u) &= Q_\alpha(\{\|GCN(u, \Theta^*_{\epsilon,i})\|_2 - err_i | \forall i \in \mathcal{V}_{\text{train}} \setminus \{u\}\}) \\ \mathbb{C}^+_{\Theta^*}(u) &= Q_{1-\alpha}(\{\|GCN(u, \Theta^*_{\epsilon,i})\|_2 + err_i | \forall i \in \mathcal{V}_{\text{train}} \setminus \{u\}\}) \end{aligned} \tag{7}$$

where Q_α and $Q_{1-\alpha}$ are the α and $(1-\alpha)$ quantile of a set. Since a wide confidence interval of node u means that the model is less confident with respect to node u, it implies that node u has high uncertainty. Following this intuition, the uncertainty of node u can be naturally quantified by the width of the corresponding confidence interval (Eq. (7)). Since the uncertainty is quantified using the confidence interval constructed by a jackknife estimator, we term it as *jackknife uncertainty* which is formally defined in Definition 1.

Definition 1. *(Node Jackknife Uncertainty) Given an input graph $\mathcal{G}$ with node set $\mathcal{V}$, a set of training nodes $\mathcal{V}_{train} \subseteq \mathcal{V}$ and an L-layer GCN with parameters Θ, $\forall i \in \mathcal{V}_{train}$ and $\forall u \in \mathcal{V}$, we assume (1) the nodes are exchangeable, and denote that (2) the LOO parameters are $\Theta_{\epsilon,i}$, (3) the error is defined as Eq.* (6) *and (4) the lower bound $\mathbb{C}^-_\Theta(u)$ and the upper bound $\mathbb{C}^+_\Theta(u)$ of predictive confidence interval are defined as Eq.* (7)*, the jackknife uncertainty of node u is*

$$\mathbb{U}_\Theta(u) = \mathbb{C}^+_\Theta(u) - \mathbb{C}^-_\Theta(u) \tag{8}$$

We note that Alaa and van Der Schaar [2] leverage high-order influence functions to quantify the jackknife uncertainty for IID data. Though Eq. (4) shares the same form as in [2] when the order is up to 1, our work bears three subtle differences. First, [2] views the model parameters as statistical functionals of data distribution and exploits von Mises expansion over the data distribution to estimate the LOO parameters,[3] which is fundamentally different from our Taylor expansion-based estimation. Specifically, von Mises expansion requires that the perturbed data distribution should be in a convex set of the original data distribution and all possible empirical distributions [16]. Since the node distribution of a graph is often unknown, the basic assumption of von Mises expansion might not hold on graph data. However, our definition relies on the Taylor expansion over model parameters which are often drawn

[2]A sequence of random variables is exchangeable if and only if the joint distribution of the random variables remains unchanged regardless of their ordering in the sequence [44]. This assumption is commonly used in random graph models, e.g., Erdős-Rényi model [14], stochastic block model [22].

[3]A statistical functional is a map that maps a distribution to a real number.

independently from Gaussian distribution(s). Thus, our method is able to generalize on graphs. Second, [2] works for regression or binary classification tasks by default, whereas we target more general learning settings on graphs (e.g., multi-class node classification). Third, jackknife uncertainty is always able to quantify aleatoric uncertainty and epistemic uncertainty simultaneously on IID data. Nevertheless, as shown in Proposition 1, it requires additional assumption to quantify both types of uncertainty on GCN simultaneously for a node u.

Proposition 1. *(Necessary condition of aleatoric and epistemic uncertainty quantification on GCN) Given an input graph $\mathcal{G}$ whose node set is $\mathcal{V}$, a node $u \in \mathcal{V}$, a set of training nodes $\mathcal{V}_{train}$ and an L-layer GCN, jackknife uncertainty quantifies the aleatoric uncertainty and the epistemic uncertainty as long as u is outside the L-hop neighborhood of an arbitrary training node $v \in \mathcal{V}_{train} \setminus \{u\}$.*

Proof. See Appendix. □

Remark. For GCN, jackknife uncertainty cannot always measure the aleatoric uncertainty and epistemic uncertainty simultaneously. In fact, if a node u is one of the neighbors within L hops with respect to any training node $v \in \mathcal{V}_{\text{train}} \setminus \{u\}$, jackknife uncertainty only quantifies the epistemic uncertainty due to lack of knowledge on the loss of node u. In this case, jackknife uncertainty cannot quantify aleatoric uncertainty because leaving out the loss of node u does not necessarily remove node u in the graph. More specifically, the aleatoric uncertainty of node u can still be transferred to its neighbors through neighborhood aggregation in graph convolution.

3.2 The Influence Functions of GCN

In order to quantify jackknife uncertainty (Eq. (8)), we need to compute the influence functions to estimate the leave-one-out parameters by Eq. (4). Given a GCN with Θ being the set of model parameters, to compute the influence functions of node i (Eq. (5)), we need to compute two key terms, including (1) first-order derivative $\nabla_{\Theta} r(i, \mathbf{y}_i, \Theta)$ and (2) second-order derivative $\mathbf{H}_{\Theta}$. We first give the following proposition for the computation of first-order derivative in Proposition 2.[4] Based on that, we present the main results for computing the second-order derivative in Theorem 1. Finally, we show details of influence function computation in Algorithm 1.

Proposition 2. *(First-order derivative of GCN [25]) Given an L-layer GCN whose parameters are Θ, an input graph $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$, a node i with its label $\mathbf{y}_i$ and a node-wise loss function $r(i, \mathbf{y}_i, \Theta)$ for node i, the first-order derivative of loss function $r(i, \mathbf{y}_i, \Theta)$ with respect to the parameters $\mathbf{W}^{(l)}$ in the l-th graph convolution layer is*

$$\nabla_{\mathbf{W}^{(l)}} r(i, \mathbf{y}_i, \Theta) = \left(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\right)^T \left(\frac{\partial r(i, \mathbf{y}_i, \Theta)}{\partial \mathbf{E}^{(l)}} \circ \sigma'(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)})\right) \tag{9}$$

where $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is the renormalized graph Laplacian with $\tilde{\mathbf{D}}$ being the degree matrix of $\mathbf{A} + \mathbf{I}$, σ' is the derivative of the activation function σ, $\circ$ is the element-wise product, and $\frac{\partial r(i, \mathbf{y}_i, \Theta)}{\partial \mathbf{E}^{(l)}}$ can be iteratively calculated by

$$\frac{\partial r(i, \mathbf{y}_i, \Theta)}{\partial \mathbf{E}^{(l)}} = \hat{\mathbf{A}}^T \left(\frac{\partial r(i, \mathbf{y}_i, \Theta)}{\partial \mathbf{E}^{(l+1)}} \circ \sigma'(\hat{\mathbf{A}}\mathbf{E}^{(l)}\mathbf{W}^{(l+1)})\right)\left(\mathbf{W}^{(l+1)}\right)^T \tag{10}$$

Proof. See Appendix. □

Since the parameters are often represented as matrices in the hidden layers, the second-order derivative will be a 4-dimensional tensor (i.e., a Hessian tensor). Building upon the results in Proposition 2, we first present the computation of the Hessian tensor in Theorem 1. Then we discuss efficient computation of influence function, which is summarized in Algorithm 1.

Theorem 1. *(The Hessian tensor of GCN) Following the settings of Proposition 2, denoting the overall loss $R(\mathcal{G}, \mathcal{Y}_{train}, \Theta)$ as R and σ'_l as $\sigma'(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)})$, the Hessian tensor $\mathfrak{H}_{l,i} = \frac{\partial^2 R}{\partial \mathbf{W}^{(l)} \partial \mathbf{W}^{(i)}}$ of R with respect to $\mathbf{W}^{(l)}$ and $\mathbf{W}^{(i)}$ has the following forms.*

Case 1. *$i = l$, $\mathfrak{H}_{l,i} = 0$*

Case 2. *$i = l - 1$*

$$\mathfrak{H}_{l,i}[:, :, c, d] = \left(\hat{\mathbf{A}} \frac{\partial \mathbf{E}^{(l-1)}}{\partial \mathbf{W}^{(i)}[c, d]}\right)^T \left(\frac{\partial R}{\partial \mathbf{E}^{(l)}} \circ \sigma'_l\right) \tag{11}$$

where $\frac{\partial \mathbf{E}^{(l-1)}}{\partial \mathbf{W}^{(i)}[c,d]}$ is the matrix whose entry at the a-th row and the b-th column is

$$\frac{\partial \mathbf{E}^{(l-1)}[a, b]}{\partial \mathbf{W}^{(l-1)}[c, d]} = \sigma'_{l-1}[a, b]\left(\hat{\mathbf{A}}\mathbf{E}^{(l-2)}\right)[a, c]\mathbf{I}[b, d] \tag{12}$$

Case 3. *$i < l - 1$*

- *Apply Eq. (12) for the i-th hidden layer.*
- *Forward to the $(l-1)$-th layer iteratively with*

$$\frac{\partial \mathbf{E}^{(l-1)}}{\partial \mathbf{W}^{(i)}[c, d]} = \sigma'_{l-1} \circ \left(\hat{\mathbf{A}} \frac{\partial \mathbf{E}^{(l-2)}}{\partial \mathbf{W}^{(i)}[c, d]} \mathbf{W}^{(l-1)}\right) \tag{13}$$

- *Apply Eq. (11).*

Case 4. *$i = l + 1$*

$$\mathfrak{H}_{l,i}[:, :, c, d] = (\hat{\mathbf{A}}\mathbf{E}^{(l-1)})^T \left(\frac{\partial^2 R}{\partial \mathbf{E}^{(l)} \partial \mathbf{W}^{(i)}[c, d]} \circ \sigma'_l\right) \tag{14}$$

where $\frac{\partial^2 R}{\partial \mathbf{E}^{(l)}[a,b] \partial \mathbf{W}^{(l+1)}[c,d]} = \mathbf{I}[b, c]\left[\hat{\mathbf{A}}^T\left(\frac{\partial R}{\partial \mathbf{E}^{(l+1)}} \circ \sigma'_{l+1}\right)\right][a, d]$.

Case 5. *$i > l + 1$*

- *Compute $\frac{\partial^2 R}{\partial \mathbf{E}^{(i-1)} \partial \mathbf{W}^{(i)}[c,d]}$ whose (a, b)-th entry has the form $\frac{\partial^2 R}{\partial \mathbf{E}^{(i-1)}[a,b] \partial \mathbf{W}^{(i)}[c,d]} = \mathbf{I}[b, c]\left(\hat{\mathbf{A}}^T\left(\frac{\partial R}{\partial \mathbf{E}^{(i)}} \circ \sigma'_i\right)\right)[a, d]$*
- *Backward to $(l+1)$-th layer iteratively with*

$$\frac{\partial^2 R}{\partial \mathbf{E}^{(l)} \partial \mathbf{W}^{(i)}[c, d]} = \hat{\mathbf{A}}^T \left(\frac{\partial^2 R}{\partial \mathbf{E}^{(l+1)} \partial \mathbf{W}^{(i)}[c, d]} \circ \sigma'_{l+1}\right)\left(\mathbf{W}^{(l+1)}\right)^T \tag{15}$$

- *Apply Eq. (14).*

Proof. See Appendix. □

Even with Proposition 2 and Theorem 1, it is still non-trivial to compute the influence of node u due to (C1) the high-dimensional nature of the Hessian tensor and (C2) the high computational cost of Eq. (5) due to the inverse operation. Regarding the first challenge (C1), for any node u, we observe that each element in the first-order derivative $\nabla_{\mathbf{W}^{(l)}} R$ is the element-wise first-order derivative, i.e., $\nabla_{\mathbf{W}^{(l)}} R[a, b] = \frac{\partial R}{\partial \mathbf{W}^{(l)}[a,b]}$. Likewise, for the Hessian tensor, we have $\mathfrak{H}_{l,i}[a, b, c, d] = \frac{\partial^2 R}{\partial \mathbf{W}^{(l)}[a,b] \partial \mathbf{W}^{i}[c,d]}$.[5] Thus, the key idea to solve the first challenge (C1) is to vectorize the first-order derivative into a column vector and compute the element-wise second-order derivatives accordingly, which naturally flatten the Hessian tensor

[4]The method to compute the first-order derivative was first proposed in [25] for a different purpose, i.e., ensuring degree-related fairness in GCN.

[5]We use R to represent $R(\mathcal{G}, \mathcal{Y}_{\text{train}}, \Theta)$ for notational simplicity.

into a Hessian matrix. More specifically, we first vectorize the first-order derivatives of R with respect to $\mathbf{W}^{(l)}, \forall l \in \{1, \ldots, L\}$ in to column vectors and stack them vertically as follows,

$$\mathbf{f}_R = \begin{bmatrix} \mathrm{vec}(\nabla_{\mathbf{W}^{(1)}} R) = \mathrm{vec}(\frac{\partial R}{\partial \mathbf{W}^{(1)}}) \\ \vdots \\ \mathrm{vec}(\nabla_{\mathbf{W}^{(L)}} R) = \mathrm{vec}(\frac{\partial R}{\partial \mathbf{W}^{(L)}}) \end{bmatrix} \quad (16)$$

where vec() vectorizes a matrix to a column vector. Then the flattened Hessian matrix is a matrix $\mathbf{H}_{\text{flat}}$ whose rows are of the form

$$\mathbf{H}_{\text{flat}}[(i \cdot c + d), :] = \left(\frac{\partial \mathbf{f}_R}{\partial \mathbf{W}^{(i)}[c, d]}\right)^T = \mathrm{vec}(\mathfrak{H}_{l,i}[:, :, c, d])^T \quad (17)$$

Finally, we follow the strategy to compute the influence of node u: (1) Compute $\nabla_{\mathbf{W}^{(l)}} r(u, \mathbf{y}_u, \Theta)$ for all l-th hidden layer; (2) Vectorize $\nabla_{\mathbf{W}^{(l)}} r(u, \mathbf{y}_u, \Theta)$ and stack to column vector $\mathbf{f_u}$ as shown in Eq. (16); (3) Compute the influence function $\mathbb{I}(u) = \mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u$.

Regarding the second challenge (C2), the key idea is to apply Hessian-vector product (Algorithm 1) [2, 28], which approximates $\mathbb{I}(u) = \mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u$ using the power method. Mathematically, it treats $\mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u$ as one vector and iteratively computes

$$\mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u = \mathbf{f}_u + (\mathbf{I} - \hat{\mathbf{H}}_{\text{flat}})(\mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u) \quad (18)$$

where $\hat{\mathbf{H}}_{\text{flat}}(\mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u)$ is viewed as a vector and $\hat{\mathbf{H}}_{\text{flat}}$ is the flattened Hessian matrix with respect to a set of sampled nodes at current iteration. The workflow of the Hessian-vector product is presented in Algorithm 1. For any l-th hidden layer (step 2), we first compute the first-order derivative $\nabla_{\mathbf{W}^{(l)}} r(u, \mathbf{y}_u, \Theta)$ with respect to $\mathbf{W}^{(l)}$ (step 3). Then we vectorize it to a column vector and stack it to $\mathbf{f}_u$ that stores the first-order derivatives of all hidden layers (step 4). After all first-order derivatives are computed, we apply the power method to compute the Hessian-vector product. In each iteration, we first sample a batch of t training nodes, which helps reduce both noise and running time, and then compute the empirical loss over these nodes (steps 7 – 8). After that, we compute the second-order derivatives with Theorem 1 and flatten it to a matrix with the strategy shown in Eq. (17) (step 9). We finish this iteration by computing Eq. (18) (step 10). The power method (steps 7 – 10) iterates until the maximum number of iteration is reached to ensure the convergence. Consequently, Algorithm 1 offers a computationally friendly way to approximate influence functions without involving both tensor-level operations and the computationally expensive matrix inversion.

Remark. We observe that both the first-order derivative (Proposition 2) and the second-order derivative (Theorem 1) can be computed in the style of neighborhood aggregation. Due to the well-known over-smoothness of GCN and the homophily nature of neighborhood aggregation, the resulting influence functions by Algorithm 1 may follow the homophily principle as well. For two nodes under homophily, due to similarity between their influences, their corresponding LOO parameters and LOO errors could be similar as well, which in turn could cause similar uncertainty scores by Definition 1. The potential homophily phenomenon in jackknife uncertainty is consistent with the homophily assumption with respect to uncertainty/confidence in existing works [40, 47, 49].

Algorithm 1: Hessian-Vector Product

```
Input :An input graph $\mathcal{G}$, training nodes $\mathcal{V}_{\text{train}}$,
       ground-truth labels $\mathcal{Y}_{\text{train}}$, node $u$ with label $\mathbf{y}_u$, an
       $L$-layer GCN with parameters $\Theta$, a node-wise loss
       function $r$, sampling batch size $t$, #iterations $m$;
Output:The influence $\mathbb{I}_\Theta(u)$ of node $u$.
1  Initialize $\mathbf{f}_u = []$ as an empty column vector;
2  for $l = 1 \to L$ do
3      Compute $\nabla_{\mathbf{W}^{(l)}} r(u, \mathbf{y}_u, \Theta)$ by Eq. (9);
4      Vectorize $\nabla_{\mathbf{W}^{(l)}} r(u, \mathbf{y}_u, \Theta)$ and stack it to $\mathbf{f}_u$ as Eq. (16);
5  Initialize $(\mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u)_0 \leftarrow \mathbf{f}_u$;
6  for $iter = 1 \to m$ do
7      Uniformly sample $t$ training nodes and get $\mathcal{V}_s$;
8      Compute empirical loss $R_s \leftarrow \frac{1}{|\mathcal{V}_s|}\sum_{i \in \mathcal{V}_s} r(i, \mathbf{y}_i, \Theta)$;
9      Compute $\hat{\mathbf{H}}_{\text{flat}}$ of $R_s$ with Theorem 1 and Eq. (17);
10     Compute $(\mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u)_{iter} \leftarrow \mathbf{f}_u + (\mathbf{I} - \hat{\mathbf{H}}_{\text{flat}})(\mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u)_{iter-1}$
11 return $(\mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u)_m$;
```

4 JURYGCN: ALGORITHM AND APPLICATIONS

In this section, we present our proposed method named JuryGCN to quantify node jackknife uncertainty (Algorithm 2) followed by discussions on applications and generalizations of JuryGCN.

4.1 JuryGCN Algorithm

With Algorithm 1, the LOO parameters of each node can be efficiently computed by proper initialization on the perturbation coefficient (ϵ in Eq. (5)). After that, the LOO predictions and LOO errors can be efficiently inferred by simply switching the original parameters to the LOO parameters, resulting in efficient jackknife uncertainty quantification.

Based on that, Algorithm 2 presents the general workflow of our proposed JuryGCN to quantify the jackknife uncertainty of a node. In detail, with proper initialization (step 1), we loop through each training node i to quantify their influences (step 2). For each training node i, it estimates the LOO parameters by leaving out training node i (steps 3 – 6), outputs the LOO predictions of nodes i and u (step 7) and compute the LOO error of each training node i (step 8). After the LOO predictions of node u and the LOO errors of all training nodes are obtained, we compute the lower bound and upper bound of the predictive confidence interval (steps 9 – 10). Finally, the uncertainty of the node is computed as the width of the predictive confidence interval (step 11).

4.2 JuryGCN Applications

After quantifying the jackknife uncertainty of each node, we utilize the node uncertainty in (1) active learning on node classification and (2) semi-supervised node classification. The details of uncertainty-aware active learning and node classification are as follows.

Application # 1: Active Learning on Node Classification. In general, active learning sequentially selects a subset of data points to query according to an acquisition function, which is designed to identify the most informative samples, and hence improves the model performance from the obtained labels. In active learning on node classification, we are given (1) an unlabelled training set of nodes (i.e., $\mathcal{V}_{\text{train}}$), (2) a node classifier (e.g., GCN), (3) step size b, and (4) the query budget K. At each query step, according to

```
Algorithm 2: JuryGCN: Jackknife Uncertainty Quantification
Input  : An input graph $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$ with training nodes
         $\mathcal{V}_{\text{train}}$, a node $u$, a GCN with parameters $\Theta$, a
         node-wise loss function $r$, a coverage parameter $\alpha$;
Output : The uncertainty $\mathbb{U}_\Theta(u)$ of node $u$.
1  Initialize $\epsilon \leftarrow -\frac{1}{|\mathcal{V}_{\text{train}}|}$;
2  for $i \in \mathcal{V}_{train}$ do
3      Compute node-wise loss $r_{i,\Theta} \leftarrow r(i, \mathbf{y}_i, \Theta)$;
4      Compute node-wise derivative $\nabla_\Theta r_{i,\Theta}$;
5      Compute $\mathbb{I}_\Theta(i) \leftarrow \mathbf{H}_\Theta^{-1} \nabla_\Theta r_{i,\Theta}$ using Algorithm 1;
6      Estimate LOO model parameters $\Theta_{\epsilon,i} \leftarrow \Theta + \epsilon \mathbb{I}_\Theta(i)$;
7      Output LOO predictions $GCN(i, \Theta_{\epsilon,i})$ and $GCN(u, \Theta_{\epsilon,i})$;
8      Compute LOO error $err_i \leftarrow \|\mathbf{y}_i - GCN(i, \Theta_{\epsilon,i})\|_2$;
9  Compute lower bound
   $\mathbb{C}_\Theta^-(u) \leftarrow Q_\alpha(\{\|GCN(u, \Theta_{\epsilon,i})\|_2 - err_i | i \in \mathcal{V}_{\text{train}} \setminus \{u\}\})$;
10 Compute upper bound
   $\mathbb{C}_\Theta^+(u) \leftarrow Q_{1-\alpha}(\{\|GCN(u, \Theta_{\epsilon,i})\|_2 + err_i | i \in \mathcal{V}_{\text{train}} \setminus \{u\}\})$;
11 return $\mathbb{U}_\Theta(u) \leftarrow \mathbb{C}_\Theta^+(u) - \mathbb{C}_\Theta^-(u)$;
```

the acquisition function, we select b nodes from the remaining unlabelled nodes in the training set $\mathcal{V}_{\text{train}}$ to query and then re-train the classifier. The query step is repeated until the query budget K is exhausted. Intuitively, a node with high predictive uncertainty is a better query candidate compared to the one with certain prediction. From Algorithm 2, we can obtain the jackknife uncertainty of each node in $\mathcal{V}_{\text{train}}$, hence, we define the acquisition function as follows, $\text{Acq}(\mathcal{V}_{\text{train}}) = \text{argmax}_{u \in \mathcal{V}_{\text{train}}} \mathbb{U}_\Theta(u)$, where $\mathbb{U}_\Theta(u)$ is the jackknife uncertainty of node u from the remaining unlabelled nodes in $\mathcal{V}_{\text{train}}$ and is computed in Eq. (8). Therefore, at each step, we select b unlabelled nodes with the top-b jackknife uncertainty. The detailed experimental settings are introduced in Appendix.

Application # 2: Semi-supervised Node Classification. In training a GCN-based node classification model, existing approaches treat each training node equally and compute the mean of loss from all training nodes, i.e., $R = \frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{i \in \mathcal{V}_{\text{train}}} r(i, \mathbf{y}_i, \Theta)$ where the node-specific loss (i.e., $r(\cdot)$) is factored by an identical weight (i.e., $\frac{1}{|\mathcal{V}_{\text{train}}|}$). Intuitively, training nodes have various levels of uncertainty during training, the easily predicted samples (i.e., small uncertainty) may comprise the majority of the total loss and hence dominate the gradient, which makes the training inefficient. To address this problem, based on jackknife uncertainty estimation, we introduce a dynamic scale factor, β, to adjust the importance for nodes with different levels of uncertainty. Specifically, given the cross-entropy loss that is utilized in semi-supervised node classification, we define the uncertainty-aware node-specific loss as, $r_u = -\beta_u^\tau \log(p_u^{(i)})$ where $p_u^{(i)}$ is the predictive probability of the i-th class from GCN and τ is a hyperparameter. The scale factor of node u is computed by normalizing the uncertainty over all training nodes, i.e., $\beta_u = \frac{|\mathbb{U}_\Theta(u)|}{\sqrt{\sum_{i \in \mathcal{V}_{\text{train}}} |\mathbb{U}_\Theta(i)|^2}}$. By introducing β_u, the loss of node with larger uncertainty (i.e., $\mathbb{U}_\Theta$) would be upweighted in the total loss and hence the training is guided toward nodes with high uncertainty. Therefore, when training a node classification model, we leverage Algorithm 2 to estimate the jackknife uncertainty of the training nodes and then apply the obtained uncertainty results to update the loss every few number of epochs.

In addition to Tasks 1 and 2, our proposed JuryGCN is generalizable to other learning tasks and graph mining models. Due to the space limit, we only present brief descriptions of each generalization direction, which could be a future direction of JuryGCN.

Applications beyond Node Classification. The proposed JuryGCN can be applied to a variety of learning tasks. For example, it can estimate the confidence interval of the predictive dependent variable in a regression task by replacing the cross-entropy loss for node classification with mean squared error (MSE) [2]. Besides active learning, reinforcement learning (RL) is extensively explored to model the interaction between agent and environment. The proposed framework is applicable to RL in the following two aspects. First, in the early stage of training an RL model, the uncertainty quantification results can be utilized to guide exploration. Second, through effectively quantifying the uncertainty of the reward after taking certain actions, JuryGCN can also benefit the exploitation. Specifically, as a classic topic of RL, multi-armed bandit can also be a potential application for JuryGCN where uncertainty is highly correlated with decision making.

Beyond JuryGCN: Generalizations to Other GNN Models. The proposed JuryGCN is able to be generalized to other graph mining models with the help of automatic differentiation in many existing packages, e.g., PyTorch, TensorFlow. In this way, only model parameters and the loss function are required in computing the first-order and second-order derivatives for influence function computation (Algorithm 1), which is further used in Algorithm 2 for jackknife uncertainty quantification.

5 EXPERIMENTAL EVALUATION

In this section, we conduct experiments to answer the following research questions:

RQ1. How effective is JuryGCN in improving GCN predictions?
RQ2. How efficient is JuryGCN in terms of time and space?
RQ3. How sensitive is JuryGCN to hyperparameters?

5.1 Experimental Settings

1 – Datasets. We adopt four widely used benchmark datasets, including Cora [38], Citeseer [38], Pubmed [35] and Reddit [20]. Table 1 summarizes the statistics of the datasets.

Table 1: Statistics of datasets.

Datasets	Cora	Citeseer	PubMed	Reddit
# nodes	2, 708	3, 327	19, 717	232, 965
# edges	5, 429	4, 732	44, 338	114, 615, 892
# features	1, 433	3, 703	500	602
# classes	7	6	3	41

2 – Comparison Methods. We compare the proposed algorithm with node classification methods in the following two categories, including (1) active learning-based approaches, which aim to select the most informative nodes to query, and (2) semi-supervised methods. In active learning on node classification, we include the following methods for comparison, AGE [5], ANRMAB [18], Coreset [39], Centrality, Random and SOPT-GCN [36]. For semi-supervised node classification, we have two uncertainty-based approaches, including S-GNN [49] and GPN [40], as well as GCN [27] and GAT [43]. For detailed description of the baselines, please refer to Appendix.

3 – Evaluation Metric. In this paper, we use Micro-F1 to evaluate the effectiveness. In terms of efficiency, we compare the running time (in seconds) and the memory usage (in MB).
4 – Implementation Details. We introduce the implementation details of the experiments in Appendix.
5 – Reproducibility. All datasets are publicly available. All codes are programmed in Python 3.6.9 and PyTorch 1.4.0. All experiments are performed on a Linux server with 96 Intel Xeon Gold 6240R CPUs and 4 Nvidia Tesla V100 SXM2 GPUs with 32 GB memory.

5.2 Effectiveness Results (RQ1)

1 – Active Learning on Node Classification. Table 2 presents the results of active learning at different query steps. We highlight the best performing approach in bold and underline the best competing one respectively. We have the following observations: **(1)** At the final query step (i.e., the 5-th line for each dataset), JuryGCN selects more valuable training nodes than other baseline methods, resulting in higher Micro-F1. Though AGE outperforms JuryGCN on *Pubmed* at 40 queries, the superiority is very marginal (i.e., 0.1%); **(2)** In general, JuryGCN achieves a much better performance when the query size is small. For example, on *Cora* dataset, JuryGCN outperforms SOPT-GCN by 2.3% at 20 queries while the gain becomes 0.6% at 80 queries. One possible explanation is that newly-selected query nodes may contain less fruitful information about new classes/features for a GCN classifier that is already trained with a certain number of labels; **(3)** As the query size increases, the gained performance of each method is diminishing, which is consistent with the second observation. For instance, on *Reddit* dataset, ANRMAB improves 13.5% when query size becomes 100 (from 50), while the gain is only 1.5% at 250 queries.
2 – Semi-supervised Node Classification. We classify nodes by training with various numbers of training nodes. Figure 1 summarizes the results on four datasets. We can see that, in general, under the conventional setting of semi-supervised node classification (i.e., the right most bar where the number of training nodes is similar with [27, 43]), JuryGCN improves the performance of GCN to certain extent w.r.t. Micro-F1 (dark blue vs. yellow). Nonetheless, GCN can achieve a slightly better performance than JuryGCN on *Cora*. In the mean time, as the number of training nodes becomes smaller, we can observe a consistent superiority of the proposed JuryGCN over other methods. For example, on *Citeseer*, when 20 training labels are provided, JuryGCN outperforms the best competing method (i.e., GPN) by 2.3% w.r.t. Micro-F1, which further demonstrate the effectiveness of JuryGCN in quantifying uncertainty when the training labels are sparse.

To summarize, the uncertainty obtained by the proposed JuryGCN is mostly valuable when either the total query budget (for active learning) or the total available labels (for semi-supervised node classification) is small. This is of high-importance especially for high-stake applications (e.g., medical image segmentation) where the cost of obtaining high-quality labels is high.

5.3 Efficiency Results (RQ2)

We evaluate the efficiency of JuryGCN on *Reddit* in terms of running time and memory usage (Figure 2). Regarding running time, we compare the influence function-based estimation with re-training when leaving out one sample at a time. In Figure 2a, as the number of training nodes increases, the total running time of re-training becomes significantly larger than that of JuryGCN. For example, JuryGCN can achieve over 130× speed-up over re-training with $10,000$ training labels. In terms of memory usage in Figure 2b, compared to GPN and S-GNN, JuryGCN (i.e., blue diamond at the upper-left corner) reaches the best balance between Micro-F1 and memory usage, with the best effectiveness and low memory usage.

5.4 Parameter and Sensitivity Analysis (RQ3)

We investigate the sensitivity of JuryGCN w.r.t. (1) the coverage parameter α in active learning at the third query step and (2) the hyperparameter τ in semi-supervised node classification, where the number of training labels corresponds to the third value on x-axis in Figure 1. Figure 3 presents the results of sensitivity analysis. For active learning on node classification (i.e., Figure 3a), the Micro-F1 results represent the performance at the third query step (i.e., the middle line of each dataset in Table 2). And for semi-supervised classification, the number of labels corresponds to the third value on x-axis in Figure 1. Regarding the sensitivity of α, results in Figure 3a shows that Micro-F1 slightly decreases as α increases. It might because smaller α indicates a larger target coverage $(1-2\alpha)$[6], resulting in wider confidence intervals. Hence, different levels of uncertainty can be accurately captured for selecting valuable query nodes. Regarding the sensitivity of τ, we can observe from Figure 3b that JuryGCN is comparatively robust to τ from 0 (i.e., cross-entropy loss) to 3. Meanwhile, by adjusting the importance of training node with the scale factor, Micro-F1 of JuryGCN improves, which is consistent with our findings in Section 5.2.

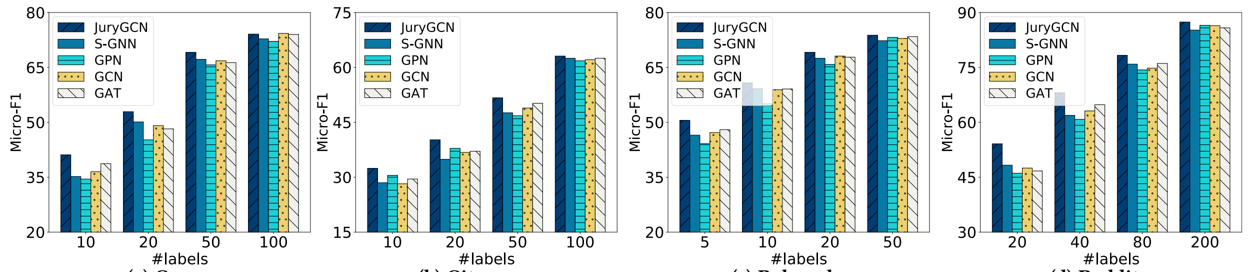
6 RELATED WORK

Graph neural networks (GNNs) often reveal the state-of-the-art empirical performance on many tasks like classification [27], anomaly detection [12] and recommendation [46, 51]. Bruna et al. [4] leverages a learnable diagonal matrix to simulate the convolution operation in graph signal processing. Defferrard et al. [11] improve the efficieny of convolution operation on graphs with the Chebyshev expansion of the graph Laplacian. Kipf and Welling [27] approximates the spectral graph convolution with neighborhood aggregation over one-hop neighbors. Hamilton et al. [20] inductively learn node representations by sampling and aggregating node representations over the local neighborhood of a node. Veličković et al. [43] introduce the self-attention mechanism to graph neural networks. Chen et al. [8] enable batch training on GCN by sampling the receptive fields in each hidden layer. Rong et al. [37] drops a certain number of edges during each training epoch of a graph neural network. Different from [8, 37] that utlizes dropout or sampling based strategies to learn node representations with uncertainty, our work deterministically quantifies the uncertainty in a post-hoc manner. Wang et al. [47] calibrate the confidence of GCN by utilizing another GCN as the calibration function. Our work differs from [47] that we do not rely on any additional model to learn the confidence interval of GCN. We refer to recent survey [50] for more related works on GNNs.
Uncertainty quantification aims to understand to what extent a model is likely to misclassify a data sample. There has been a rich collection of research works in quantifying uncertainty for IID data [2, 17, 29, 32, 33, 42]. More related works on IID data can be

[6]The theoretical coverage of jackknife+ is $1-2\alpha$.

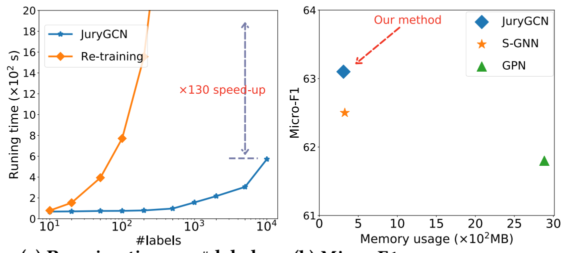
Table 2: Performance comparison results on active learning on node classification w.r.t. Micro-F1. Higher is better.

Data	Query size	JuryGCN (Ours)	ANRMAB	AGE	Coreset	Centrality	Degree	Random	SOPT-GCN
Cora	20	**51.1 ± 1.2**	46.8 ± 0.5	<u>49.4 ± 1.0</u>	43.8 ± 0.8	41.9 ± 0.6	38.5 ± 0.7	40.5 ± 1.6	48.8 ± 0.7
	40	**64.7 ± 0.8**	61.2 ± 0.8	58.2 ± 0.7	55.4 ± 0.5	57.3 ± 0.7	48.4 ± 0.3	56.8 ± 1.3	<u>62.6 ± 0.8</u>
	60	**69.9 ± 0.9**	67.8 ± 0.7	65.7 ± 0.8	62.2 ± 0.6	63.1 ± 0.5	58.8 ± 0.6	64.5 ± 1.5	<u>67.9 ± 0.6</u>
	80	**74.2 ± 0.7**	73.3 ± 0.6	72.5 ± 0.4	70.2 ± 0.5	69.1 ± 0.4	67.6 ± 0.4	69.7 ± 1.6	<u>73.6 ± 0.5</u>
	100	**75.5 ± 0.6**	74.9 ± 0.4	74.2 ± 0.3	73.8 ± 0.4	74.1 ± 0.3	73.0 ± 0.2	74.2 ± 1.2	**75.5 ± 0.7**
Citeseer	20	**38.4 ± 1.5**	35.9 ± 1.0	33.1 ± 0.9	30.2 ± 1.2	35.6 ± 1.1	31.5 ± 0.9	30.3 ± 2.3	<u>36.1 ± 0.7</u>
	40	**51.1 ± 0.9**	46.7 ± 1.3	49.5 ± 0.6	42.1 ± 0.8	<u>49.8 ± 1.3</u>	39.8 ± 0.7	41.1 ± 1.8	49.2 ± 0.5
	60	**58.2 ± 0.8**	55.2 ± 0.9	56.1 ± 0.5	52.1 ± 0.9	<u>57.1 ± 0.7</u>	50.1 ± 1.1	49.8 ± 1.3	56.4 ± 0.5
	80	**63.8 ± 1.1**	<u>63.2 ± 0.7</u>	61.5 ± 0.8	59.9 ± 0.6	63.3 ± 1.0	58.8 ± 0.6	58.1 ± 1.1	63.2 ± 0.8
	100	**64.3 ± 1.2**	<u>64.1 ± 0.5</u>	63.2 ± 0.7	62.8 ± 0.4	63.9 ± 0.6	61.8 ± 0.5	62.9 ± 0.8	63.8 ± 0.6
Pubmed	10	**61.8 ± 0.9**	60.5 ± 1.3	58.9 ± 1.1	53.1 ± 0.7	55.8 ± 1.2	56.4 ± 1.5	52.4 ± 1.7	59.5 ± 0.6
	20	**70.2 ± 0.6**	66.8 ± 1.1	<u>68.7 ± 0.7</u>	62.8 ± 0.5	67.2 ± 1.4	64.3 ± 1.0	60.5 ± 1.4	67.9 ± 0.9
	30	**73.9 ± 0.3**	71.6 ± 0.8	<u>72.8 ± 1.0</u>	68.9 ± 0.3	73.5 ± 0.9	70.1 ± 0.7	68.9 ± 1.1	72.3 ± 0.8
	40	<u>74.6 ± 0.4</u>	73.2 ± 0.6	**74.7 ± 0.8**	72.8 ± 0.8	74.1 ± 0.7	72.0 ± 0.8	71.8 ± 1.2	73.8 ± 0.7
	50	**75.4 ± 0.5**	74.7 ± 0.4	75.1 ± 0.5	73.5 ± 0.6	74.2 ± 0.6	72.9 ± 0.5	73.1 ± 1.0	<u>75.2 ± 0.5</u>
Reddit	50	**69.7 ± 1.7**	67.8 ± 0.9	64.2 ± 1.1	62.1 ± 0.6	65.5 ± 1.2	62.5 ± 1.4	63.7 ± 2.4	<u>68.1 ± 1.2</u>
	100	**82.9 ± 1.5**	<u>81.3 ± 1.0</u>	79.5 ± 0.8	81.2 ± 1.0	78.2 ± 0.9	81.1 ± 1.2	80.5 ± 1.6	80.4 ± 1.3
	150	**86.0 ± 1.4**	84.3 ± 0.7	83.2 ± 0.4	84.8 ± 0.9	84.1 ± 1.1	82.5 ± 1.2	81.5 ± 1.4	<u>85.0 ± 1.5</u>
	200	**88.1 ± 0.9**	86.1 ± 0.8	85.8 ± 0.5	85.5 ± 0.8	87.5 ± 0.8	85.4 ± 0.7	83.1 ± 1.8	<u>87.2 ± 0.9</u>
	250	**89.2 ± 0.8**	87.6 ± 0.7	87.1 ± 0.4	86.6 ± 1.1	<u>88.7 ± 0.6</u>	86.1 ± 1.0	87.3 ± 1.5	87.8 ± 1.1



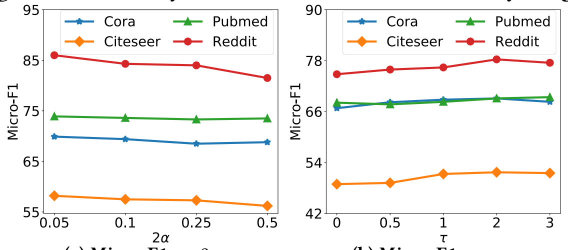
(a) Cora (b) Citeseer (c) Pubmed (d) Reddit

Figure 1: Node classification results under various numbers of labels. Higher is better. Best viewed in color.



(a) Running time vs. # labels. (b) Micro-F1 vs. memory usage.

Figure 2: Efficiency results w.r.t. time and memory usage.



(a) Micro-F1 vs. 2α. (b) Micro-F1 vs. τ.

Figure 3: Parameter study on the coverage parameter α and the hyperparameter τ.

found in recent survey [1]. Regarding uncertainty quantification for graph data, Dallachiesa et al. [10] classify nodes in consideration of uncertainty in edge existence. Hu et al. [23] learns node embeddings for an undirected uncertain graph. Eswaran et al. [15] use Dirichlet distribution to model uncertainty in belief propagation. Ng et al. [36] propose Graph Gaussian Process (GGP) as a Bayesian linear model on the feature maps of nodes. Liu et al. [30] further extend GGP by considering input graph as an uncertain graph. Zhang et al. [48] propose Bayesian Graph Convolutional Neural Networks which infer the joint posterior of node labels given the random graph and weight parameters. Hasanzadeh et al. [21] train graph neural networks with random mask for each edge in each layer drawn from a Bernoulli distribution. Different from [10, 15, 21, 23, 48], our work deterministically quantifies the uncertainty using influence functions. Zhao et al. [49] propose Subjective GNN (S-GNN) to model uncertainty of graph neural networks in both deep learning and subjective logic domain. Stadler et al. [40] propose Graph Posterior Netwokr, which extends the Posterior Network [6] to graphs for uncertainty estimation. Compared with [40, 49], our work quantifies uncertainty in a post-hoc manner without changing the training procedures of GCN.

7 CONCLUSION

In this paper, we study the problem of jackknife uncertainty quantification on Graph Convolutional Network (GCN) from the frequentist perspective. We formally define the jackknife uncertainty of a node as the width of confidence interval by a jackknife (leave-one-out) estimator. To scale up the computation, we rely on the influence functions for efficient estimation of the leave-one-out parameters without re-training. The proposed JuryGCN framework is applied to both active learning, where the most uncertain nodes are selected to query the oracle, and semi-supervised node classification, where the jackknife uncertainty serves as the importance of loss to focus on nodes with high uncertainty. Extensive evaluations on real-world datasets demonstrate the efficacy of JuryGCN in both active learning and semi-supervised node classification. Our proposed JuryGCN is able to generalize on other learning tasks beyond GCN, which is the future direction we would like to investigate.

ACKNOWLEDGEMENT

This work is supported by National Science Foundation under grant No. 1947135, and 2134079 by the NSF Program on Fairness in AI in collaboration with Amazon under award No. 1939725, by DARPA HR001121C0165, by NIFA award 2020-67021-32799, and Army Research Office (W911NF2110088). The content of the information in this document does not necessarily reflect the position or the policy of the Government or Amazon, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

[1] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, et al. 2021. A Review of Uncertainty Quantification in Deep Learning: Techniques, Applications and Challenges. *Information Fusion* (2021).
[2] Ahmed Alaa and Mihaela van Der Schaar. 2020. Discriminative Jackknife: Quantifying Uncertainty in Deep Learning via Higher-Order Influence Functions. In *ICML*.
[3] Rina Foygel Barber, Emmanuel J Candes, Aaditya Ramdas, and Ryan J Tibshirani. 2021. Predictive Inference with the Jackknife+. *The Annals of Statistics* 49, 1 (2021), 486–507.
[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *ICLR*.
[5] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2017. Active Learning for Graph Embedding. *arXiv preprint arXiv:1705.05085* (2017).
[6] Bertrand Charpentier, Daniel Zügner, and Stephan Günnemann. 2020. Posterior Network: Uncertainty Estimation without OOD Samples via Density-Based Pseudo-Counts. In *NeurIPS*. 1356–1367.
[7] Cen Chen, Kenli Li, Sin G Teo, Xiaofeng Zou, Kang Wang, Jie Wang, and Zeng Zeng. 2019. Gated Residual Recurrent Graph Neural Networks for Traffic Prediction. In *AAAI*. 485–492.
[8] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*.
[9] R Dennis Cook and Sanford Weisberg. 1982. *Residuals and Influence in Regression*. New York: Chapman and Hall.
[10] Michele Dallachiesa, Charu Aggarwal, and Themis Palpanas. 2014. Node Classification in Uncertain Graphs. In *SSDBM*. 1–4.
[11] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*.
[12] Kaize Ding, Qinghai Zhou, Hanghang Tong, and Huan Liu. 2021. Few-Shot Network Anomaly detection via cross-network meta-learning. In *WWW*.
[13] Bradley Efron. 1992. Jackknife-After-Bootstrap Standard Errors and Influence Functions. *Journal of the Royal Statistical Society: Series B (Methodological)* (1992).
[14] Paul Erdős, Alfréd Rényi, et al. 1960. On the Evolution of Random Graphs. *Publ. Math. Inst. Hung. Acad. Sci* 5, 1 (1960), 17–60.
[15] Dhivya Eswaran, Stephan Günnemann, and Christos Faloutsos. 2017. The Power of Certainty: A Dirichlet-Multinomial Model for Belief Propagation. In *SDM*.
[16] Luisa Turrin Fernholz. 2012. *Von Mises Calculus for Statistical Functionals*. Vol. 19. Springer Science & Business Media.
[17] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as A Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *ICML*. 1050–1059.
[18] Li Gao, Hong Yang, Chuan Zhou, Jia Wu, Shirui Pan, and Yue Hu. 2018. Active Discriminative Network Representation Learning. In *IJCAI*. 2142–2148.
[19] Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy BR Hayter, Richard Vickers, Charles Roberts, Jian Tang, et al. 2021. Utilizing Graph Machine Learning within Drug Discovery and Development. *Briefings in Bioinformatics* 22, 6 (2021).
[20] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1025–1035.
[21] Arman Hasanzadeh, Ehsan Hajiramezanali, Shahin Boluki, Mingyuan Zhou, Nick Duffield, Krishna Narayanan, and Xiaoning Qian. 2020. Bayesian Graph Neural Networks with Adaptive Connection Sampling. In *ICML*. 4094–4104.
[22] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. 1983. Stochastic Blockmodels: First Steps. *Social Networks* 5, 2 (1983), 109–137.
[23] Jiafeng Hu, Reynold Cheng, Zhipeng Huang, Yixang Fang, and Siqiang Luo. 2017. On Embedding Uncertain Graphs. In *CIKM*. 157–166.
[24] Shengding Hu, Zheng Xiong, Meng Qu, Xingdi Yuan, Marc-Alexandre Côté, Zhiyuan Liu, and Jian Tang. 2020. Graph policy network for transferable active learning on graphs. *arXiv preprint arXiv:2006.13463* (2020).
[25] Jian Kang, Yan Zhu, Yinglong Xia, Jiebo Luo, and Hanghang Tong. 2022. RawlsGCN: Towards Rawlsian Difference Principle on Graph Convolutional Network. In *WWW*.
[26] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.
[27] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[28] Pang Wei Koh and Percy Liang. 2017. Understanding Black-Box Predictions via Influence Functions. In *ICML*. 1885–1894.
[29] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *NIPS*.
[30] Zhao-Yang Liu, Shao-Yuan Li, Songcan Chen, Yao Hu, and Sheng-Jun Huang. 2020. Uncertainty Aware Graph Gaussian Process for Semi-Supervised Learning. In *AAAI*. 4957–4964.
[31] Yifei Ma, Roman Garnett, and Jeff Schneider. 2013. Σ-Optimality for Active Learning on Gaussian Random Fields. In *NIPS*.
[32] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. 2019. A Simple Baseline for Bayesian Uncertainty in Deep Learning. In *NeurIPS*.
[33] Andrey Malinin and Mark JF Gales. 2018. Predictive Uncertainty Estimation via Prior Networks. In *NIPS*.
[34] Rupert G Miller. 1974. The Jackknife-A Review. *Biometrika* 61, 1 (1974), 1–15.
[35] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and U Edu. 2012. Query-Driven Active Surveying for Collective Classification. In *MLG*. 1.
[36] Yin Cheng Ng, Nicolò Colombo, and Ricardo Silva. 2018. Bayesian Semi-Supervised Learning with Graph Gaussian Processes. In *NIPS*.
[37] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *ICLR*.
[38] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Magazine* (2008).
[39] Ozan Sener and Silvio Savarese. 2017. Active Learning for Convolutional Neural Networks: A Core-set Approach. *arXiv preprint arXiv:1708.00489* (2017).
[40] Maximilian Stadler, Bertrand Charpentier, Simon Geisler, Daniel Zügner, and Stephan Günnemann. 2021. Graph Posterior Network: Bayesian Predictive Uncertainty for Node Classification. In *NeurIPS*.
[41] John Tukey. 1958. Bias and Confidence in Not-Quite Large Sample. *Ann. Math. Statist.* 29 (1958), 614.
[42] Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. 2020. Uncertainty Estimation Using a Single Deep Deterministic Neural Network. In *ICML*.
[43] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[44] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. 2005. *Algorithmic Learning in a Random World*. Springer Science & Business Media.
[45] Daixin Wang, Jianbin Lin, Peng Cui, Quanhui Jia, Zhen Wang, Yanming Fang, Quan Yu, Jun Zhou, Shuang Yang, and Yuan Qi. 2019. A Semi-Supervised Graph Attentive Network for Financial Fraud Detection. In *ICDM*. IEEE, 598–607.
[46] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *SIGIR*. 165–174.
[47] Xiao Wang, Hongrui Liu, Chuan Shi, and Cheng Yang. 2021. Be Confident! Towards Trustworthy Graph Neural Networks via Confidence Calibration. In *NeurIPS*.
[48] Yingxue Zhang, Soumyasundar Pal, Mark Coates, and Deniz Ustebay. 2019. Bayesian Graph Convolutional Neural Networks for Semi-supervised Classification. In *AAAI*, Vol. 33. 5829–5836.
[49] Xujiang Zhao, Feng Chen, Shu Hu, and Jin-Hee Cho. 2020. Uncertainty Aware Semi-Supervised Learning on Graph Data. In *NeurIPS*, Vol. 33.
[50] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph Neural Networks: A Review of Methods and Applications. *AI Open* 1 (2020), 57–81.
[51] Qinghai Zhou, Liangyue Li, and Hanghang Tong. 2019. Towards Real Time Team Optimization. In *IEEE BigData*.
[52] Qinghai Zhou, Liangyue Li, Xintao Wu, Nan Cao, Lei Ying, and Hanghang Tong. 2021. Attent: Active attributed network alignment. In *WWW*.

APPENDIX

Additional Experimental Settings

1 – Comparison Methods. In this section, we present the detailed description of comparison methods that are used in the experiments. For the application of active learning on node classification, we have the following methods.

- **AGE** [5] measures the informativeness of each node by considering the following perspectives, including (1) the entropy of the prediction results, (2) the centrality score, and (3) the distance between the corresponding representation and its nearest cluster center. At each step of query, nodes with the highest scores are selected.
- **ANRMAB** [18] leverages the same selection criterion as in AGE. Additionally, ANRMAB proposes a multi-armed bandit framework to dynamically adjust weights for the three perspectives. ANRMAB utilizes the performance score of previous query steps as the rewards to learn the optimal combination of weights during the query process.
- **Coreset** [39] is originally proposed for Convolutional Neural networks and performs k-means clustering on the vector representations from the last hidden layer. We follow Hu et al. [24] and apply Coreset on the node representations obtained by GCN. At each query step, we select the node which is the closet to the cluster center to label.
- **Centrality** selects the nodes with the highest scores of betweenness centrality at each query step.
- **Degree** queries the node with the highest degree.
- **Random** annotates node randomly at each query step.
- **SOPT-GCN** [36] utilizes Σ-optimal (SOPT) acquisition function as the active learner [31], which requires the graph Laplacian and the indices of labeled nodes. We follow the same setting as in [36] to conduct the experiments.

For semi-supervised node classification, we compare the proposed framework with the following approaches.

- **S-GNN** [49] is an uncertainty-aware estimation framework which leverages a graph-based kernel Dirichlet distribution to estimate different types of uncertainty associated with the prediction results. We utilize the obtained node-level representations to perform classification in the experiments.
- **GPN** [40] derives three axioms for characterizing the predictive uncertainty and proposes Graph Posterior Network (GPN) to perform Bayesian posterior updates over predictions based on density estimation and diffusion. Similarly, we utilize the node representations for classification task.
- **GCN** [27] learns node-level representations by stacking multiple layers of spectral graph convolution.
- **GAT** [43] computes the representation of each node by introducing the learnable attention weights from its neighbors.

2 – Implementation Details. In the experiment, we conduct empirical evaluations in two applications, including (1) Application #1: active learning on node classification, and (2) Application #2: semi-supervised node classification, as described in Section 4.2.

In Application #1, we evaluate all methods on a randomly constructed test set of 1,000 nodes for *Cora*, *Citeseer*, *Pubmed* datasets and 139,779 nodes for *Reddit*. The validation sets for the first three citation networks contains 500 nodes, and 23,296 is the size of validation set for *Reddit*. The remaining nodes comprise the training set (i.e., $\mathcal{V}_{\text{train}}$) where the nodes for query are selected. For *Cora*, *Citeseer*, *Pubmed* and *Reddit*, (1) we have 10, 10, 5 and 20 randomly selected labels, respectively, to initiate the computation of jackknife uncertainty using Algorithm 2; (2) the query budgets are set as 100, 100, 50 and 250; and (3) the query step sizes are 20, 20, 10 and 50.

In Application #2, we randomly selected 100, 100, 50 and 200 nodes from *Cora*, *Citeseer*, *Pubmed* and *Reddit* respectively as the training nodes. The sizes of test sets are the same as those in Application 1. During training, we run Algorithm 2 to perform uncertainty estimation over the training nodes every 10 epochs and update the scale factor α accordingly. In addition, we also evaluate the model performance when the number of training nodes is significantly small.

In both applications, we adopt a two-layer GCN with 16 hidden layer dimension. For training, we use the Adam optimizer [26] with learning rate 0.01 and train the GCN classifier for 100 epochs. The coverage parameter (α) in Algorithm 2 is 0.025 and the hyperparameter τ in Application 2 is set as 2. For all other comparison methods, we use the original settings . We report the average results after 20 runs of each method on two applications.

PROOF OF PROPOSITION 1

For an arbitrary node $v \in \mathcal{V}$ in the graph, its receptive field after L graph convolution layers is the set of all neighbors within L hops. Then if the node u, whose uncertainty is going to be quantified, is not within the L-hop neighborhood of any training node $v \in \mathcal{V}_{\text{train}} \setminus \{u\}$, whether to leave out the loss of node u during training will have no impact on hidden node representations and final predictions of v, which is equivalent to the leave-one-out settings for IID data.

PROOF OF PROPOSITION 2

To prove it, we derive the element-wise computation of $\nabla_{\mathbf{W}^{(l)}} r(i, \mathrm{y}_i, \Theta)$ and then write out the matrix form. We first apply the chain rule and get

$$\frac{\partial r(i, \mathrm{y}_i, \Theta)}{\partial \mathbf{W}^{(l)}[a,b]} = \sum_{c=1}^{n} \sum_{d=1}^{h_l} \frac{\partial r(i, \mathrm{y}_i, \Theta)}{\partial \mathbf{E}^{(l)}[c,d]} \frac{\partial \mathbf{E}^{(l)}[c,d]}{\partial \mathbf{W}^{(l)}[a,b]} \tag{19}$$

where h_l is the hidden dimension of l-th layer. Regarding the computation of $\frac{\partial \mathbf{E}^{(l)}[c,d]}{\partial \mathbf{W}^{(l)}[a,b]}$, since $\mathbf{E}^{(l)} = \sigma(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)})$, we have

$$\begin{aligned}\frac{\partial \mathbf{E}^{(l)}[c,d]}{\partial \mathbf{W}^{(l)}[a,b]} &= \frac{\partial \sigma\big(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)}\big)[c,d]}{\partial \big(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)}\big)[c,d]} \frac{\partial \big(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)}\big)[c,d]}{\partial \mathbf{W}^{(l)}[a,b]} \\ &= \sigma'\big(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)}\big)[c,d]\big(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\big)[c,a]\mathbf{I}[d,b]\end{aligned} \tag{20}$$

where σ' is the first-order derivative of the activation function σ. Combining Eqs. (19) and (20) together, we have the element-wise computation of $\nabla_{\mathbf{W}^{(l)}} r(i, \mathrm{y}_i, \Theta)$ as follows.

$$\frac{\partial r(i, \mathrm{y}_i, \Theta)}{\partial \mathbf{W}^{(l)}[a,b]} = \big(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\big)^T[a,:]\left(\frac{\partial r(i, \mathrm{y}_i, \Theta)}{\partial \mathbf{E}^{(l)}} \circ \sigma'\big(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)}\big)\right)[:,b] \tag{21}$$

where $\circ$ is the element-wise product. Finally, we get Eq. (9) by writing Eq. (21) into matrix form.

Regarding the computation of $\frac{\partial r(i,\mathrm{y}_i,\Theta)}{\partial \mathbf{E}^{(l)}}$, the key idea is to write out a recursive function with respect to $\frac{\partial r(i,\mathrm{y}_i,\Theta)}{\partial \mathbf{E}^{(l)}}$ based on the chain rule. More specifically, we first consider the element-wise computation and apply the chain rule as follows.

$$\frac{\partial r(i,\mathrm{y}_i,\Theta)}{\partial \mathbf{E}^{(l)}[c,d]} = \sum_{e=1}^{n}\sum_{f=1}^{h_{l+1}} \frac{\partial r(i,\mathrm{y}_i,\Theta)}{\partial \mathbf{E}^{(l+1)}[e,f]}\frac{\partial \mathbf{E}^{(l+1)}[e,f]}{\partial \mathbf{E}^{(l)}[c,d]} \tag{22}$$

We take derivative on both sides of $\mathbf{E}^{(l)} = \sigma(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)})$ and get

$$\frac{\partial \mathbf{E}^{(l+1)}[e,f]}{\partial \mathbf{E}^{(l)}[c,d]} = \sigma'\big(\hat{\mathbf{A}}\mathbf{E}^{(l)}\mathbf{W}^{(l+1)}\big)[e,f]\hat{\mathbf{A}}[e,c]\mathbf{W}^{(l+1)}[d,f] \tag{23}$$

Combining Eqs. (22) and (23) together, we get the following element-wise first-order derivative.

$$\begin{aligned}\frac{\partial r(i,\mathrm{y}_i,\Theta)}{\partial \mathbf{E}^{(l)}[c,d]} = \sum_{e=1}^{n}\sum_{f=1}^{h_{l+1}} \hat{\mathbf{A}}^T[c,e]\cdot\left(\frac{\partial r(i,\mathrm{y}_i,\Theta)}{\partial \mathbf{E}^{(l+1)}}\sigma'\big(\hat{\mathbf{A}}\mathbf{E}^{(l)}\mathbf{W}^{(l+1)}\big)\right)[e,f]\\ \cdot\big(\mathbf{W}^{(l+1)}\big)^T[f,d]\end{aligned} \tag{24}$$

Written Eq. (24) into matrix form, we complete the proof.

PROOF OF THEOREM 1

We prove case by case.

Case 1. When $i = l$, since the activation function σ is the ReLU function, it is trivial that the subgradient of its second-order derivative is always 0 since the first-order derivative is the indicator function. Thus, $\mathfrak{H}_{l,l} = 0$.

Case 2. When $i = l-1$, to get Eq. (11), it is trivial to prove by taking derivative on both sides of Eq. (9). See the proof of Proposition 2 for the proof of Eq. (12).

Case 3. When $i < l-1$, we first take derivative on both sides of Eq. (9) in i-th hidden layer. Then we have

$$\mathfrak{H}_{l,i}[:,:,c,d] = \left(\hat{\mathbf{A}}\frac{\partial \mathbf{E}^{(i-1)}}{\partial \mathbf{W}^{(i)}[c,d]}\right)^T\left(\frac{\partial R}{\partial \mathbf{E}^{(i)}}\circ\sigma'_l\right) \tag{25}$$

To compute $\frac{\partial \mathbf{E}^{(l-1)}}{\partial \mathbf{W}^{(i)}[c,d]}$, we first consider an arbitrary (a,b)-th element in $\frac{\partial \mathbf{E}^{(l-1)}}{\partial \mathbf{W}^{(i)}[c,d]}$, i.e., $\frac{\partial \mathbf{E}^{(l-1)}[a,b]}{\partial \mathbf{W}^{(i)}[c,d]}$. Then we take the derivative on both sides of $\frac{\partial \mathbf{E}^{(l-1)}[a,b]}{\partial \mathbf{W}^{(i)}[c,d]}$, which gives us

$$\begin{aligned}\frac{\partial \mathbf{E}^{(l-1)}[a,b]}{\partial \mathbf{W}^{(i)}[c,d]} &= \sigma'_{l-1}[a,b]\cdot\sum_{e=1}^{n}\sum_{f=1}^{h_i}\hat{\mathbf{A}}[a,e]\frac{\partial \mathbf{E}^{(l-2)}[e,f]}{\partial \mathbf{W}^{(i)}[c,d]}\mathbf{W}^{(l-1)}[f,b]\\ &= \left(\sigma'_{l-1}\circ\left(\hat{\mathbf{A}}\frac{\partial \mathbf{E}^{(l-2)}}{\partial \mathbf{W}^{(i)}[c,d]}\mathbf{W}^{(l-1)}\right)\right)[a,b]\end{aligned} \tag{26}$$

It is trivial to get Eq. (13) by writing out the matrix form of Eq. (26).

Case 4. When $i = l+1$, to get Eq. (14), it is trivial to prove by taking derivative on both sides of Eq. (9). Regarding the computation of $\frac{\partial^2 R}{\partial \mathbf{E}^{(l)}[a,b]\partial \mathbf{W}^{(l+1)}[c,d]}$, by Eq. (10), we have

$$\frac{\partial R}{\partial \mathbf{E}^{(l)}[a,b]} = \left(\hat{\mathbf{A}}^T\left(\frac{\partial R}{\partial \mathbf{E}^{(l+1)}}\circ\sigma'\big(\hat{\mathbf{A}}\mathbf{E}^{(l)}\mathbf{W}^{(l+1)}\big)\right)\big(\mathbf{W}^{(l+1)}\big)^T\right)[a,b] \tag{27}$$

Then we take derivative on both sides and get

$$\begin{aligned}\frac{\partial^2 R}{\partial \mathbf{E}^{(l)}[a,b]\partial \mathbf{W}^{(l+1)}[c,d]} &= \sum_{e=1}^{n}\sum_{f=1}^{h_{l+1}}\hat{\mathbf{A}}^T[a,e]\cdot\left(\frac{\partial R}{\partial \mathbf{E}^{(l+1)}}\sigma'_{l+1}\right)[e,f]\\ &\qquad\cdot\frac{\partial \mathbf{W}^{(l+1)}[b,f]}{\partial \mathbf{W}^{(l+1)}[c,d]}\\ &= \mathbf{I}[b,c]\sum_{e=1}^{n}\hat{\mathbf{A}}^T[a,e]\cdot\left(\frac{\partial R}{\partial \mathbf{E}^{(l+1)}}\sigma'_{l+1}\right)[e,d]\\ &= \mathbf{I}[b,c]\left(\hat{\mathbf{A}}^T\left(\frac{\partial R}{\partial \mathbf{E}^{(l+1)}}\circ\sigma'_{l+1}\right)\right)[a,d]\end{aligned} \tag{28}$$

Case 5. When $i > l+1$, we get Eq. (15) by taking derivative on both sides of Eq. (10).

Putting everything (Cases 1 – 5) together, we complete the proof.