Attention-Augmented Parametric Kernel Graph Neural Network (APKGNN) for Node Classification

Avishek Bose
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
bosea@ornl.gov

William H. Hsu
Department of Computer Science and Engineering
Kansas State University
Manhattan, Kansas, USA
bhsu@ksu.edu

2023 International Conference on Machine Learning and Applications (ICMLA) | 979-8-3503-4534-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICMLA58977.2023.00343

***Abstract*—We present a new graph neural network, the Attention-based Parametric-Kernel augmented Graph Neural Network (APKGNN), developed for node classification tasks. Despite extensive work on modeling multi-faceted relationships between connected nodes of a graph, the effect of attention on edge features mapped to relationships has not yet been analyzed through learning representation. This study derives such an attention vector by first calculating node features corresponding to endpoints of an edge and then aggregating these with extracted local intrinsic patches of a given graph to generate augmented local patch vectors. This process uses a parametric kernel based on Gaussian mixture models (GMMs) to embed local neighborhoods of the graph in local patches. The patch vectors then convolve with the above node features to produce an updated node representation. We show that this new learning representation (APKGNN) achieves higher node classification accuracy on tasks - both standard benchmarks (Cora, PubMed, Citeseer) and new experimental short text corpora where nodes correspond to text documents and words. This implementation of the GNN convolution layer outperforms state-of-the-art (SOTA) algorithms, achieving higher training, validation, and test accuracy by a significant margin on three standard benchmark data sets under both SOTA experimental settings and those for new testbeds.**

***Index Terms*—APKGNN, parametric kernel, attention vector, local patch, neighborhood sampling, multi-processing**

I. Introduction

Despite being successfully applied to many real-life applications, such as image classification [1], object detection [2], [3], machine translation [4], and speech recognition [5], Convolutional Neural Network (CNN) [6] and Recurrent Neural Network (RNN) [7] cannot be directly applied to generic graph structure data especially in non-Euclidean domains such as social networks [8], High Performance Computing (HPC) Analytics [9], telecommunication networks, and epidemic networks because of their structural limitations. For the non-Euclidean domain, information from both properties of a graph, (i) the intrinsic features of nodes and (ii) the relationships between the nodes, is important. To learn information from both types of sources for a downstream predictive task such as node classification, Graph Neural Networks (GNN) are a learning representation technique where the predictive result is calculated by neighborhood information aggregation. For many node classification tasks, semi-supervised learning approaches like GNNs outperform CNN models for which less labeled data limits learning models.

Existing GNN layers such as Monet [10] has a common problem in extracting expressive local intrinsic patches (also known as receptive fields) because, in it, a node feature attention is not considered by its neighbors. From a different direction, in an attention-based method such as GAT [11], edge attributes do not convolve with a parametric kernel to produce updated patch vectors resulting in less expressive local patches. This approach generates aggregated node feature representation without applying a parametric kernel (e.g., Gaussian mixture model) through computing a scalar attention score for each edge. The scalar value combines all low dimensional information of each edge together which limits the discriminative power of kernel operation. Consequently, this approach leads to average performing learning representation for the downstream tasks. On the other hand, using a fixed kernel for neighborhood information aggregation as in GCN [12] also shows limitations in learning real-world complex and large graph structures to produce expressive node representations.

For instance, a relationship between two humans (humans considered as nodes) depends on such things as their natures, favorite items, origins, interests, and food habits, and related elements like communication, proximity, and common social media platform. In the same way, an edge connection between two nodes having a single weight does not encode a graph structure efficiently in the latent space. Again, each edge feature is not equally important because they represent individual features with different objectives. So, a model aggregating two sub-module computations according to the number of specified kernels is expected to be effective in learning a representation where the first one attends over the distribution of connected node features and the second one generates patches by convolving a parametric kernel over the edge features. In our proposed method, the local intrinsic patches are extracted after a parametric kernel is applied to edge attributes (pseudo-coordinates) where the result is linearly combined with edge attention vectors (from node features) to form augmented local patches. Finally, the resulting augmented local patches convolve with node features as a template-matching process between augmented local patches and features of nodes in a graph.

The limitations of earlier approaches mentioned above paragraphs become crucial when GNNs are applied to graphs of

This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript or allow others to do so, for the US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan)

1946-0759/23/$31.00 ©2023 IEEE
DOI 10.1109/ICMLA58977.2023.00343

short text corpora because the graph structure itself, by default, lacks contextual information, and any form of information gap can impact the low-level feature expressiveness such as node classification accuracy. Moreover, the cost of labeling raw text data is expensive. In this work, we demonstrate the performance of our proposed method not only on benchmark node data sets but also on a text data set named CTI data set [13](Social media corpora containing raw tweets for cyber threat type classification) where GNNs as semi-supervised learning techniques are applied to a text classification task.

This study contributes to (i) implementing a novel GNN architecture and (ii) the GNN's application to node classification tasks. The contributions are listed below:

- Implemented attention-based parametric kernel augmented GNN (APKGNN) layer architecture to implement a robust GNN model;
- Evaluated the implemented model on benchmark graph data sets, which gained performance over all of the SOTA models by a significant margin;
- Designed a GNN model building pipeline by applying all significant GNN layers (in literature) where the pipeline results show the models outperform even their corresponding original implementations.
- Developed a multi-processing model training testbed to train all the GNN models.
- Applied the implemented GNN model on a raw text data set (CTI data set) for cyber-threat type classification where the implemented model outperforms all SOTA models.

II. Related Work

The advent of graph neural network (GNN) was first inspired by recurrent neural networks applied to directed acyclic graphs Frasconi et al. (1998) [14] and Sperduti et al. (1997) [15]. Later GNN was introduced by Gori et al. (2005) [16] which was then extended by Scarselli et al. (2009) [15] and further improved by Li et al. (2016) [17] to handle general types of graphs. These methods recursively exchanged neighborhood information as a propagation method until a stable equilibrium was reached. Semi-supervised learning was first introduced in graph learning using a graph Laplacian regularization approach that includes research work such as label propagation by Zhu et al. (2003) [18], label spreading by Zhou et al. (2003) [19], manifold regularization by Belkin et al. (2006) [20], semi-supervised embedding by Weston et al. (2012) [21] applied to attribute graphs. Considering structural correlation among data samples, the non-spectral technique of graph convolution was introduced in Duvenaud et al. (2015) [22] and GraphSage Hamilton et al. (2017) [23] for graph-level classification, and in Atwood & Towsley (2016) [24] for node classification. These approaches require learning weight matrices either for each node degree or for both input channel and neighborhood degree (using the power of transition matrix) and do not scale to large graphs with a wide range of node degree distributions. Niepert et al. (2016) [25] requires normalizing fixed neighborhoods of a graph before converting them locally into ordered node sequences to feed into a convolutional neural network. Spectral graph convolutional networks inspired by graph Fourier analysis that assumes filter is a set of learnable parameters and considers graph signal with multiple channels was first introduced by Bruna et al. (2014) [26]. After that, Henaff et al. (2015) [27] parameterized spectral filters to make them spatially localized using smoothing coefficients. In a follow-up, Defferrard et al. (2016) [28] extended the research by approximating the filters using Chebyshev expansion of the graph Laplacian through avoiding computation of Laplacian eigenvectors to yield spatially localized convolutions. After that, Kipf et al. (2017) [12] simplified the ChebNet approximation using filters restricted to operating up to the first-order neighborhood of each node. An extension of this GCN was introduced by Klicpera et al. (2019) [29], improving GCN's over-smoothing by adopting the PageRank algorithm. Unlike GCN which assumes equal contributions of neighboring nodes by assigning non-parametric weights to the edges, GAT (Velickovic et al. (2018) [11]), inspired by the attention mechanism (Bahdanau et al. (2015)) [30] that could handle variable size neighborhoods, attended highly important parts of the inputs to learn the relative weights between two connected nodes. A significant, parametric kernel-based approach Monet proposed by Monti et al. (2017) [10] assigned weights to a node's neighbors by mapping with relative positions of nodes within a neighborhood by introducing a feature called node pseudo-coordinates.

III. Deep Learning on Graphs

A. Spectral and Spatial based approaches

GNN architectures are theoretically categorized into two types of approaches: spectral approaches and spatial approaches. Spectral graph convolution [26], [27] originated from network signal spectral analysis where a signal $\mathrm{X} \in \mathbb{R}^F$ or a node's set of input channel $X^{in} = (x_1^{in}, ..., x_p^{in})$ is multiplied with a parameterized filter G_θ in Fourier domain:

$$g_\theta * \mathbf{X} = \mathbf{V} G_\theta \mathbf{V}^T \mathbf{X} = \mathbf{V} G_\theta(\Lambda) \mathbf{V}^T X \tag{1}$$

where $\mathbf{V}$ is the eigenvector matrix of the normalized graph Laplacian $\mathbf{L}$, θ is the set of learnable parameters, and $\mathbb{R}$ is the set of all real numbers. The normalized graph Laplacian matrix can be represented as follows:

$$\mathbf{L} = I_N - \hat{\mathbf{D}}^{-1/2} \mathbf{A} \hat{\mathbf{D}}^{-1/2} = \mathbf{V} \Lambda \mathbf{V}^T \tag{2}$$

where $\mathbf{V}^T\mathbf{X}$ is the Fourier transform of graph signal $\mathbf{X}$, Λ is a diagonal eigenvalue matrix, and G_θ is a function of eigenvalues $G_{l,j} = diag(g_{l,j,1}, ...g_{l,j,k})$ with learnable parameters of normalized graph Laplacian $\mathbf{L}$, $\hat{\mathbf{D}}$ is normalized diagonal matrix, $\mathbf{A}$ is the adjacency matrix, and $\mathbf{I}$ is the identity matrix.

Some major drawbacks of the basic spectral approaches are that the filter depends on a given graph and the learned filter cannot be applied to different graphs; computationally expensive Fourier transformation; and filters cannot guarantee spatial consistency. A spectral approach simplified by the higher order Chebyshev polynomial by ChebyNet [28], the

convolution of the input signal $\mathbf{X}$ with a filter is represented as

$$\begin{aligned} g_\theta * \mathbf{X} &= V(\Sigma_{p=0}^{K}\theta_p T_p(\Lambda))V^T X \\ &\sim \Sigma_{p=0}^{K}\theta_p T_p(L')X \end{aligned} \tag{3}$$

where $T_p(x)$ is the recursively defined truncated expansion of the Chebyshev polynomials, K is the order of polynomials, and $\mathbf{L}'$ is the normalized Laplacian.

On the other hand, spatial GNN approaches [22], [23], [24], [25] rely on the spatial information of a node: its location in a graph, its neighbors, and its degrees, for learning graph representation where the convolution operation propagates node information along the edge connections. The spatial filter can work with variable-sized neighborhoods and can be generalized across other domains. Unlike earlier spatial-based approaches that had fixed filter operations such as [12], [29], etc. to extract local intrinsic patches on the graph, spatial approaches formulating patch operators as a function of pseudo-coordinates [10] perform better for the complex node classification task. The general form of a spatial GNN can be written as follows:

$$\mathbf{H}^i = f(\mathbf{X}\mathbf{W}^i + \sigma_{i=1}^{i-1}\mathbf{A}\mathbf{H}^{i-1}\theta^i) \tag{4}$$

where $\mathbf{H}$ is the hidden layer feature matrix, and $\mathbf{W}$ represents the linear transformation weight matrix, f is the nonlinear activation function.

Our implemented method is a spatial-based approach inspired by the ideas of the parametric kernel and attention mechanism. Now, we delve into the details of the original work to analyze their limitations and then harness their potential to address the model's inexpressiveness.

B. Parametric Kernel for Extracting Patches

However, a parametric kernel [10] can significantly improve the performance of node classification of spatial GNNs if the patch operator can be represented by Gaussian mixture model (GMM) kernels. Parametric kernel function (weight function) $(w_\theta(u) = (w_1(u), ..., w_k(u)))$ can encode graph spatial information into low dimensional space where the dimension depends on the number of kernels applied to the d-dimensional vector in general (here d=2) of pseudo-coordinates u, formally written as $u(x, y)$ known as edge attributes:

$$u(x,y) = (\frac{1}{\sqrt{deg(x)}}, \frac{1}{\sqrt{deg(y)}}) \tag{5}$$

Here x is a node in a graph, and y is the neighbor of x (e.g., $y \in \mathcal{N}(x)$), $deg()$ function returns the node degree of a given node x, θ is learnable parameters, and the number of kernels k is the dimensionality of the extracted patch. The patch operator can therefore be written in the following general form:

$$D_k(x)f = \Sigma_{y\in\mathcal{N}(x)} w_k u(x,y) f(y) \quad [k = 1, 2, ...\mathbb{N}] \tag{6}$$

On the other hand, the parametric kernel function (weight function) is defined as follows (adopted from [10]):

$$w_\theta(u) = exp(-1/2(u-\mu_k)^T \Sigma_k^{-1}(u-\mu_k)) \tag{7}$$

where μ represents the mean, and Σ represents the covariance matrix of the GMM.

C. Self Attention in GNN

Attention mechanisms [30] applied to graph neural networks [11] allow the representation of a central node to attend to its most important nodes in the neighborhood. A graph attention network (GAT) as a spatial-based approach relies on the locality of nodes, so for a given node, this aggregates neighboring node features $\mathbf{X}$ $(\{\overrightarrow{x_1}, \overrightarrow{x_2}, ..., \overrightarrow{x_N}\}, \overrightarrow{x_i} \in \mathbb{R}^F)$ after attending them. Here $\mathcal{N}$ is the number of nodes, and F is the number of node features. Each GAT layer produces a new node feature vector $\mathbf{X}$ $(\{\overrightarrow{x_1'}, \overrightarrow{x_2'}, ..., \overrightarrow{x_N'}\}, \overrightarrow{x_i'} \in \mathbb{R}^{F'})$ after a learnable linear transformation is computed by a weight matrix $\mathbf{W} \in \mathbb{R}^{F\times F'}$. To calculate the importance of neighboring node j features to node i ($j \in \mathcal{N}_i$), the concept of attention coefficients has been introduced:

$$e_{ij} = a(\mathbf{W}\overrightarrow{x_i}, \mathbf{W}\overrightarrow{x_j}) \tag{8}$$

Here, the attention function a is a single-layer, feed-forward neural network that applies LeakyRelu non-linearity parameterized by a weight vector $\overrightarrow{\mathrm{a}} \in \mathbb{R}^{2F'}$. Thus, Equation 8 can be rewritten:

$$a(\mathbf{W}\overrightarrow{x_i}, \mathbf{W}\overrightarrow{x_j}) = LeakyReLU(\overrightarrow{\mathrm{a}}^T[\mathbf{W}\overrightarrow{X}_i || \mathbf{W}\overrightarrow{X}_j]) \tag{9}$$

where $(.^T)$ refers to matrix transposition, $\mathbf{W}$ refers to the linear transformation weight matrix, and $||$ represents the concatenation of vectors. To scale the different coefficient values over each node's neighborhood, the coefficients are normalized as follows:

$$\alpha_{ij} = softmax(e_{ij}) = exp(e_{ij})/\Sigma_{q\in\mathcal{N}} exp(e_{iq}). \tag{10}$$

Attending the combined information of node features and edge attributes (without convolving with the parametric kernel) results in an imperfect learning representation for a central node because the linear combination of the two sources of information (node features and edge attributes) generates a scalar value that does not effectively project into an encoded representation according to the specified number of kernels. In contrast, conventional parametric kernel-based approaches do not take attention into account. Therefore, GNN architecture must improve representation learning by adopting the useful features of these two techniques. Moreover, unlike parametric kernel techniques, the attention model uses node features for computing similarity without considering node structural properties. This reveals a research gap: not knowing the graph structure upfront for the classification task. Hence, the limited classification accuracy in many SOTA approaches is a motivational standpoint for us to investigate further for a hybrid GNN model to address the discussed limitations.

IV. Implemented Technique

We first describe implemented APKGNN convolutional layer, the layer structure used to implement the GNN model for node classification. We have two different inputs to this layer:

1) A set of node features, $\mathbf{X} = (\{\overrightarrow{x_1}, \overrightarrow{x_2}, ..., \overrightarrow{x_F}\}, \overrightarrow{x_i} \in \mathbb{R}^{\mathcal{N}})$
2) A set of neighboring node d-dimensional pseudo-coordinates $u(x, y)$ defined as $u \in \mathcal{E}_i$, where $\mathcal{E}$ is the set of all edge connection.

From these two sets of inputs, the matrix of edge attribute vectors are represented for convolution as $\mathbf{E} = (d \times \mathcal{E}) \cdot k$, d is the dimension of edge attributes (here d=2), k is the kernel size (also known as the dimensionality of the extracted patch) described in the following Section IV, $\mathcal{N}$ is the number of nodes, F is the number of node features, x is the given node, and $y \in \mathcal{N}(x)$ is the neighboring node of x.

We first self-attend neighboring node features (features of local patches) X of a central node x, so we can define which features contribute more in learning a better graph representation. This formulation can be written in a vector representation as follows (adopted from [11]):

$$\begin{aligned}\overrightarrow{att\_vec} &= softmax(LeakyReLU([\mathbf{W}_k \overrightarrow{x}_i || \mathbf{W}_k \overrightarrow{x}_j])) \\ &= \frac{exp(LeakyReLU([\mathbf{W}_k \overrightarrow{x}_i || \mathbf{W}_k \overrightarrow{x}_j]))}{\sum_{l \in \mathcal{N}(x)} exp(LeakyReLU([\mathbf{W}_k \overrightarrow{x}_i || \mathbf{W}_k \overrightarrow{x}_l]))}\end{aligned} \tag{11}$$

where $\overrightarrow{att\_vec}$ is the attention vector for each node in their corresponding neighborhoods, dimension is defined according to the number of specified kernels k, and $||$ is the concatenation operation.

We adopted the GMM-based parametric kernel function to extract patches from neighboring edge attributes (pseudo coordinates) in a locally encoded graph structure where the kernel function (weight function) maps relative positions (pseudo coordinates) to weight vectors. The size vectors are defined according to the number of kernels also known as kernel size k. Therefore, the weight function applied to the edge attributes (or the relative positions or pseudo coordinates) is written from subsection III-B as follows:

$$w_\theta(u) = (w_1(u), ..., w_k(u)) \tag{12}$$

where θ refers to the kernel learnable parameters used to extract patches $\mathbf{P}$ from edge attribute matrix $\mathbf{E}$. The following equation derived from **Equation 7** presents GMM operation on edge attributes to extract patches $\overrightarrow{p}$

$$\overrightarrow{p} = exp(-1/2(u - \mu_k)^T \Sigma_k^{-1}(u - \mu_k))). \tag{13}$$

The set of attention vectors is aggregated with the set of extracted patches after mapping these with respect to the number of kernels k. The resulting feature attended patches are then transformed by a parameter vector (a element-wise multiplication) as follows $\overrightarrow{q}$.

$$\overrightarrow{s} = \overrightarrow{q} \cdot (\overrightarrow{att\_vec} + \overrightarrow{p}) \tag{14}$$

Now, considering the feature attended patch vector $\overrightarrow{s}$ for all the nodes we can write it as a matrix $\mathbf{S}$. Finally, the set of augmented patch vectors is convolved with the neighboring node features to compute node representations as the final output followed by a linear activation as follows:

$$\mathbf{X}^l = \sigma(\sum_{x \in \mathcal{N}(x)} \mathbf{S} * \mathbf{W}\mathbf{X}^{l-1}). \tag{15}$$

where l stands for convolution layers and $l \in \mathbb{N}$.

A two-layer APKGNN model is implemented for semi-supervised node classification on graph data sets. After applying all these steps, the forward model expression can be defined as follows:

$$\mathbf{Z} = f(E, \boldsymbol{\theta}, \mathbf{X}) = softmax(\mathbf{W}_{\theta_2} \odot ELU(\mathbf{W}_{\theta_1} \odot \mathbf{X})) \tag{16}$$

where Z is the final convolved feature matrix, $\boldsymbol{\theta}$ is set of all parameters, $\odot$ is convolution operation, and $\mathbf{W}_{\theta_1}$ and $\mathbf{W}_{\theta_2}$ are the weight matrices for first and second convolution layers respectively.

For semi-supervised multi-class classification, the cross-entropy loss over the labeled example is defined as follows:

$$\mathcal{L} = -\sum_{l \in Y_L} \sum_{f=1}^{C} Y_{lf} ln Z_{lf} \tag{17}$$

where Y_L is the set of nodes that have labels and C is the set of output classes. The implemented GNN model is trained by mini-batch gradient descent on a full graph data set for some training epochs. The mini-batches are sampled by the neighborhood node sampler technique that was proposed in the GraphSage [23] algorithm. Figure 1 demonstrates the operational steps of node classification using a graph neural network constructed by the novel APKGNN layer. This figure exhibits how mini batches split a graph into multiple sub-graphs and then a group of sub-graphs is processed in a GPU multiprocessing environment. Figure 2 displays the construction of the APKGNN layer. The horizontal line divides the figure into two processes where part A portrays parametric kernel function operation to extract local patches. Part B portrays the operation of vector attention on edge attributes.

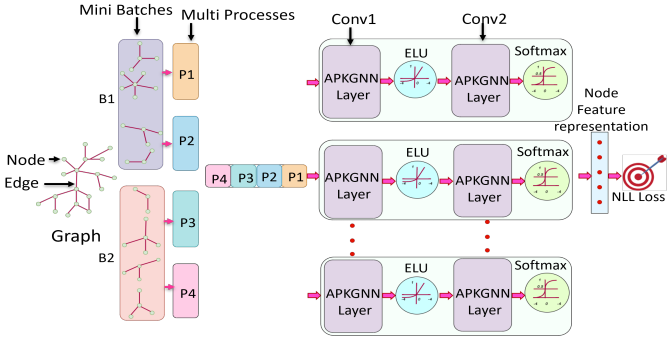


Fig. 1: Operational steps of node classification using novel implemented APKGNN layer

V. Experiments

The implemented models have been experimented on several graph benchmark data sets (citation networks) and CTI text data set for semi-supervised node classification.

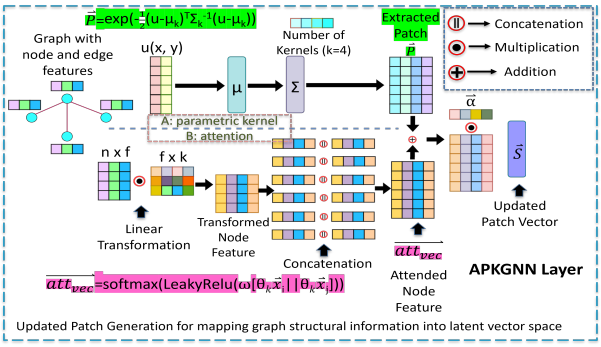


Fig. 2: Construction of APKGNN layer's principle step

A. Data sets

We followed the Yang et al. [31] experimental setup for benchmark data sets (see Table I) where we used the same benchmark data split as provided by the PyTorch Geometric [32] library. Each benchmark data set (Cora, PubMed, Citeseer) is a citation network of scientific publications where each publication is represented by a 0/1-valued word vector corresponding to the respective word dictionary. Properties of each data set consist of a number of nodes, links, labels, and features where the number of publications maps to graph nodes, publication classes map to labels, unique words from respective dictionary map to features, and citation relation map to links, respectively. Train mask, test mask, and validation mask from Table II denote node splits used for training, testing, and validation, respectively. The SOTA GNN models are applied, including the implemented GNN model APKGNN on the CTI data [13] for cyber-threat classification where the text categories of the CTI data set are mapped to different types of such cyber-threats as 0day, malware, and botnet. The CTI graph data set is split with a measurement such that the split process retains optimal information of the original text data set.

TABLE I: Data set statistics for benchmark data sets and CTI data

Dataset	Type	Nodes	links	Label	Features
Cora	Citation network	2,708	5,429	7	1,433
CiteSeer	Citation network	3312	4732	6	3703
Pubmed	Citation network	19,717	44,338	3	500
CTI	Text graph	23113	303074	10	8114

TABLE II: Training statistics for benchmark data sets and CTI data

Dataset	Training Mask	Test Mask	Validation Mask
Cora	140	1000	500
CiteSeer	120	1000	500
Pubmed	60	1000	500
CTI	5000	2500	614

B. Experimental Setup

We conducted the experiments as transductive learning tasks where we trained a two-layer APKGNN model as described in Section IV and displayed in Figure 1. The following implemented experiments were conducted on GPU multi-process environment (2 GPU servers of 4 cores each) with a neighborhood sampling method to address the need for processing big graph data sets. We evaluated prediction accuracy on the nodes of the test mask and validation mask of the benchmark data sets for hyperparameter optimization. For all the citation network data sets (Cora, Pubmed, and Citeseer) and CTI text data sets, we used equal hyperparameter settings and trained all models for 50 epochs (training iterations). We kept the default dropout rate for both two layers with p=0.5, a stable (L2 regularization) with $\lambda = 5 \times 10^{-4}$, and the number of hidden units ($|\mathbb{H}|$) as 16. We used the Adam optimizer with a learning rate lr=0.001 and an early stop with a window size of 10. The first layer of the implemented GNN network was followed by an exponential linear unit (ELU) [33] nonlinearity whereas the second layer followed by a softmax activation was used for node classification that computes C features (C is the number of classes). Although We tried multiple kernel sizes such as 16,8, and 4, it has a very slight impact on the result. So, we kept the lowest kernel size of 4 for simplicity to report our result.

All the model and network parameters, including transformation weight matrices, were initialized using the method described in this work [34]. For the CTI data, the experiment settings required slight changes because the data set was derived from raw tweets and the text graph was generated following the method described in TextGCN [35]. In this data set, document nodes did not have any direct connection; document nodes were instead connected by common word nodes shared by two different documents. In addition, only document nodes have labels (cyber-threat types) because words cannot have text categories or more specifically, cyber-threat types.

VI. Results and Performance Evaluation

The experimental results for the developed multi-process testbed are summarized in Table III with reported mean test and validation accuracies of different SOTA GNN models on benchmark data sets and CTI data. All the models were also run on an environment setting (only train and test accuracy reported) provided by a SOTA model [36] which achieved the highest accuracy reported in Table IV for a fair comparison. For the first testbed, all models were run for 50 epochs with random mini-batch on training data samples using a neighborhood batch sampler adopted from the GraphSage [23] algorithm. We kept the neighborhood sample size of 10 neighbors (a small variation in neighbor size does not impact model training result) for all the data set to reduce the load computation. For the first testbed each citation benchmark data set (Cora, CiteSeer, and Pubmed) and for the CTI data, mean test accuracy, mean validation accuracy, and mean training accuracy curves were plotted with SOTA GNN models compared to the newly introduced GNN model. FIve seed numbers were used for five different simulations to run a standard APKGNN model for each data set, and an average of five accuracy scores were reported with respective standard deviation. We found that the proposed model outperformed every model in all combinations by a significant margin and converged in fewer epochs. Figure 3(a), figure 3(b), and figure 3(c) illustrate that the APKGNN model started outperforming

all SOTA models on test, validation, and training data samples of the Cora citation data set within first 50 epochs. Figure 3(d), figure 3(e), and figure 3(f) show the same trend, with the APKGNN model outperforming all SOTA models on test, validation, and training data samples from the Pubmed citation data set. Figure 3(g), figure 3(h), and 3(i) also illustrate the consistent performance of the new model on testing, training, and validation data samples of the CiteSeer citation data set as opposed to the SOTA models.All the plots are represented with corresponding standard errors via shaded areas.

Figure 3(j), figure 3(k), and figure 3(l) also follow the same trend where the experimental model outperforms the SOTA models in testing, training, and validation data samples of the CTI data. This particular result is significant because the CTI data is generated and then transformed into graph data from raw tweets. This raw data is highly complex in terms of its sparsity, poor input features, and fewer numbers of labeled nodes because word nodes are mapped with any labels. The notion of experimenting with the CTI data set is to evaluate how the introduced layer architecture and the GNN model as a whole perform on a raw text data set as compared to other models. In addition, we used one-hot encoding for node representation, but this can be easily replaced using a transformer-based embedding in the embedding layer of the constructed GNN architecture that could even generate better results.

VII. Discussion

Implementing the APKGNN model was rooted in the theory that constructing a GNN layer produces expressive learning representations to support better experimental results. The layer-wise computation of APKGNN includes two individual modules executed together. The first module calculates attention vectors of neighboring node feature similarity and the second extract local patches by applying weight function on local graph structure components called pseudo coordinates. A linear combination of the two modules followed by learning an attribute vector produces better learning representations than SOTA GNN models in node classification tasks. The running average of accuracy plotted to different curves indicates that the performance of the models is consistent across epochs. Moreover, while at the initial stage of the simulation, the new model performs on average, once the simulation proceeds toward completion, the model outperforms all the other models. The newly implemented APKGNN model learns more parameter vectors than other models, which is why each epoch for the new model takes longer than the other models. The research, when extended to short text multi-class classification tasks, also exhibits a marked improvement over the SOTA baseline. This result indicates that learning additional parameters helps the model to learn highly complex graph data sets. Now, we deliberately raise two technical questions and address the question with a reasonable response as follows:

Why an experimental setup featured with subgraph sampling in a multi-GPU processing environment is a necessary property in training a GNN model?

Answer: To the best of our knowledge, almost all the GNN models constructed from previously introduced GNN layer architectures nicely train on smaller or standard-size graphs but suffer from limited performance on extremely large-size graphs in terms of classification accuracy and training time. When those GNN models sample a large-size graph by splitting it into multiple smaller subgraphs, the expressiveness power of the models reduces significantly. However, learning additional parameters during model training can address this limitation of the inexpressive models. Consequently, a model must balance two features, scalability, and performance. Since our introduced GNN layer and constructed model incorporate both these properties, it outperforms other models in sampling and multi-GPU processing setups. Additionally, our model outperforms other models in almost all the benchmark data sets in the most commonly used experiment settings with a slightly more computation cost.

What is the rationale behind choosing GMM and why does the inclusion of a parametric kernel significantly enhance node classification performance for this purpose?

Answer: A Kernel constructed from the Gaussian Mixture Model (GMM) learns the model's parameters from a mixture of distributions to distinguish features on low-level representations. Since learning distinctive features is crucial for classification tasks, the notion of choosing a parametric kernel for learning complex graph structures is highly reasonable. Contrary to the fixed encoding generated by conventional kernel operation, the GMM kernel generates edge feature distribution on two learnable parameter vectors mean and variance resulting in dynamic encoding for local patches. In addition, by combining the attention mechanism with the GMM kernel, we are attempting to learn a sufficient number of parameters that further enhance classification performance than other approaches.

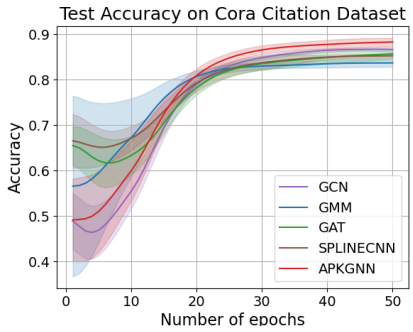
VIII. Conclusion and Future Work

This research improves node representation learning by introducing a GNN layer architecture for the node classification task. The GNN model constructed using the introduced APKGNN layer outperforms other models on all the data sets in almost all the experimental settings. The model is also trained on the CTI text data set as a semi-supervised learning application to classify different cyber-threat types. The primary testbed of the experiments is conducted in multi-processing settings with data neighborhood sampling. This study also points out what makes a graph learning model inexpressive and why it happens. We addressed these limitations by introducing a hybrid GNN layer to construct GNN models that result in higher accuracies on big graph data sets while training in GPU multi-processing setups. This analysis also leaves space for further research to optimize the number of parameters for faster training of the models.

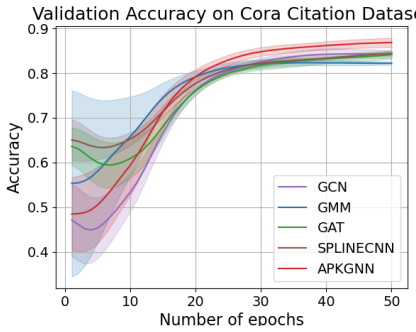
References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

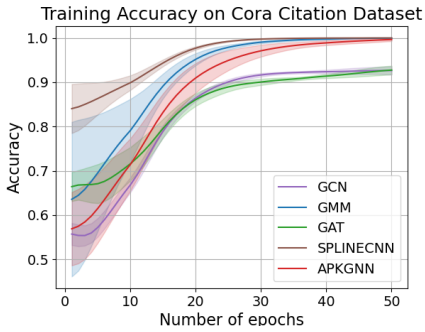
[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.



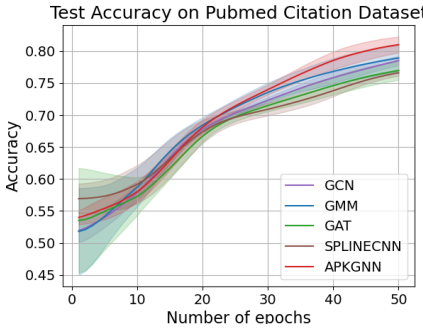
(a) Test Accuracy Curve for Cora



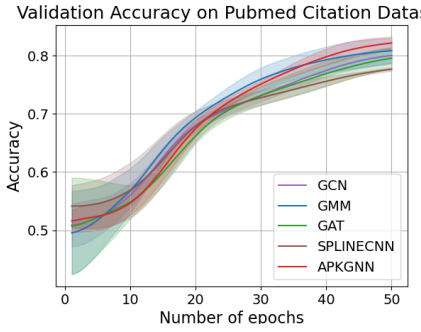
(b) Validation Accuracy Curve for Cora



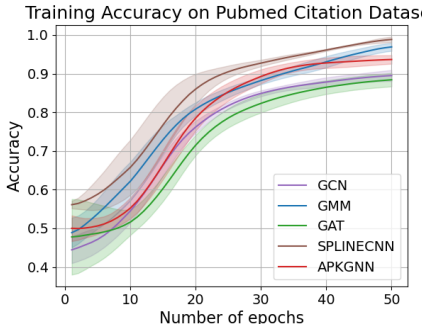
(c) Training Accuracy Curve for Cora



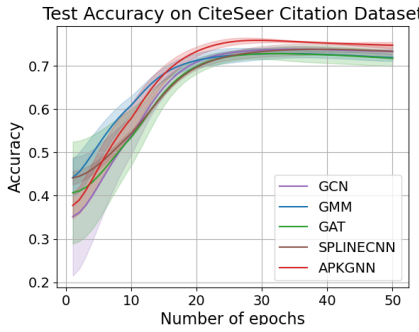
(d) Test Accuracy Curve for Pubmed



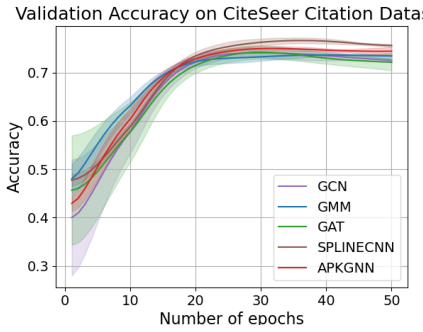
(e) Validation Accuracy Curve for Pubmed



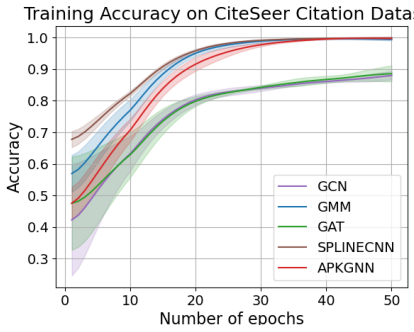
(f) Training Accuracy Curve for Pubmed



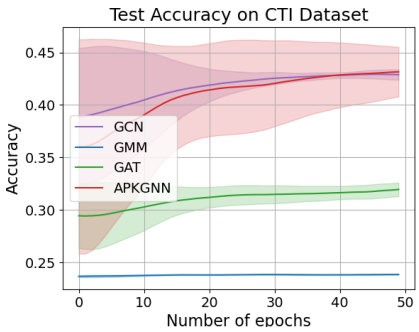
(g) Test Accuracy Curve for Citeseer



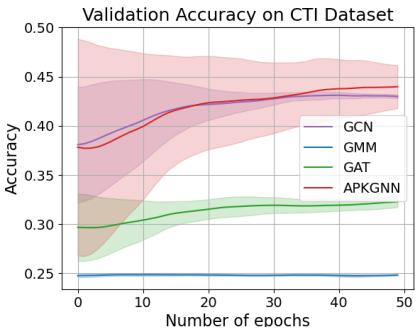
(h) Validation Accuracy Curve for Citeseer



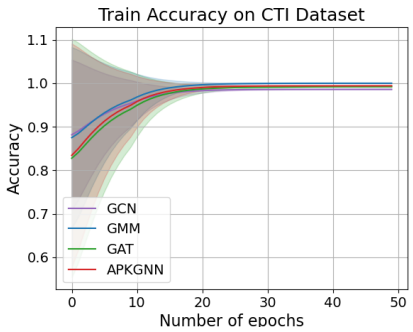
(i) Training Accuracy Curve for Citeseer



(j) Test Accuracy Curve for CTI



(k) Validation Accuracy Curve for CTI



(l) Training Accuracy Curve for CTI

Fig. 3: Test, validation, and training accuracy curves up to fifty epochs for benchmark graph and text data sets for all considered GNN models where shaded areas represent standard errors; hyperparameter settings: (p=0.5, $\lambda = 5 \times 10^{-4}$, $|\mathbb{H}|$= 16, lr=0.001 early_stop=10, k=4, epoch=50)

TABLE III: Performance of APKGNN compared to SOTA methods on validation and test accuracy for benchmark and CTI data sets based on implemented experimental set up, APKGNN results are reported with respective standard deviation

Method	Cora		Pubmed		Citeseer		CTI	
	Test	Val	Test	Val	Test	Val	Test	Val
GCN [12]	84.00	85.60	82,00	80.00	72.60	72.60	42.56	43.76
GAT [11]	84.60	85.50	81.40	79.40	72.80	73.50	32.43	33.65
GMM [10]	82.60	83.60	81.80	80.90	73.00	71.80	24.52	23.83
$SPLINECNN$ [36]	84.00	85.80	81.20	80.00	73.20	**76.00**	40.37	41.75
$APKGNN$	**86.20±0.3**	**87.50±0.3**	**82.80±0.6**	**82.40±0.6**	**74.00±0.3**	0.7540±0.3	**44.95±1.75**	**45.06±1.70**

TABLE IV: Performance of APKGNN compared to SPLINECNN on validation and test accuracy for benchmark data sets based on SPLINECNN experimental set up, APKGNN results are reported with respective standard deviation

Method	Cora		Pubmed		Citeseer	
	Test	Train	Test	Train	Test	Train
GCN [12]	79.60	96.72	85.26	85.18	75.20	97.65
GAT [11]	87.80	94.34	80.93	82.00	75.66	89.18
GMM [10]	87.60	97.38	84.33	85.15	74.60	96.65
$SPLINECNN$ [36]	88.60	**98.40**	88.20	85.20	76.60	**98.39**
$APKGNN$	**88.96±0.3**	97.54	**88.88±0.6**	**87.40**	**76.80±0.3**	0.9634

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[4] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[5] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[6] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] A. Bose, S. G. Sundari, V. Behzadan, and W. H. Hsu, "Tracing relevant twitter accounts active in cyber threat intelligence domain by exploiting content and structure of twitter network," in *2021 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2021, pp. 1–6.

[9] A. Bose, H. Yang, W. H. Hsu, and D. Andresen, "Hpcgcn: A predictive framework on high performance computing cluster log data using graph convolutional networks," in *2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 4113–4118.

[10] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5115–5124.

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

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

[13] V. Behzadan, C. Aguirre, A. Bose, and W. Hsu, "Corpus and deep learning classifier for collection of cyber threat indicators in twitter stream," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 5002–5007.

[14] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE transactions on Neural Networks*, vol. 9, no. 5, pp. 768–786, 1998.

[15] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 714–735, 1997.

[16] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE international joint conference on neural networks*, vol. 2, no. 2005, 2005, pp. 729–734.

[17] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," 2017.

[18] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of the 20th International conference on Machine learning (ICML-03)*, 2003, pp. 912–919.

[19] D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," *Advances in neural information processing systems*, vol. 16, 2003.

[20] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples." *Journal of machine learning research*, vol. 7, no. 11, 2006.

[21] J. Weston, F. Ratle, and R. Collobert, "Deep learning via semi-supervised embedding," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1168–1175.

[22] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," *Advances in neural information processing systems*, vol. 28, 2015.

[23] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[24] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[25] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*. PMLR, 2016, pp. 2014–2023.

[26] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[27] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[28] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Advances in neural information processing systems*, vol. 29, 2016.

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

[30] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[31] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *International conference on machine learning*. PMLR, 2016, pp. 40–48.

[32] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[33] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[34] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[35] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 7370–7377.

[36] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, "Splinecnn: Fast geometric deep learning with continuous b-spline kernels," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 869–877.