Journal of Network Intelligence
Taiwan Ubiquitous Information
 ISSN 2414-8105 (Online)
Volume **9**, Number **2**, May **2024**

Ensembling Graph Neural Networks for Node Classification

Ke-Ao Lin, Xiao-Zhu Xie*

College of Computer and Information Engineering
Xiamen University of Technology, Xiamen 361024, P. R. China
linkeao@foxmail.com, xz4xxz@gmail.com

Wei Weng

College of Computer and Information Engineering
Xiamen University of Technology, Xiamen 361024, P. R. China

Fujian Key Laboratory of Pattern Recognition and Image Understanding
xmutwei@163.com

Yong Chen

College of Computer and Information Engineering
Xiamen University of Technology, Xiamen 361024, P. R. China
2122031117@s.xmut.edu.cn

*Corresponding author: Xiao-Zhu Xie

Received August 23, 2023, revised November 16, 2023, accepted February 27, 2024.

Abstract. *Graph Neural Networks (GNNs) have gained considerable prominence as a formidable tool for processing graph data, such as social networks, protein structures, and chemical molecules. Node classification is a common task in GNNs that aims to predict the label or category of each node in a graph. In this paper, we propose a novel GNN ensemble model for node classification, which integrates three classical GNN models: GCN, GraphSAGE, and GAT. By integrating multiple models, we can fully leverage their advantages, improve the model's robustness and generalization ability, and reduce the risk of overfitting while achieving better performance. We conducted experiments on Cora, Citeseer, and Pubmed datasets, and the results demonstrate that our proposed ensemble model outperforms single models and other classical models in node classification tasks. This ensemble model shows great potential for application in various fields, such as social networks, bioinformatics, and recommendation systems.*
Keywords: Graph Neural Networks, Node Classification, Ensemble Model

1. **Introduction.** Graphs are a fundamental data structure that can model complex relationships and interactions in various fields, such as social networks, bioinformatics, and recommendation systems [1, 2, 3]. However, traditional neural networks, such as convolutional neural networks (CNNs) [4, 5, 6, 7] and recurrent neural networks (RNNs) [8, 9], are designed for processing grid-like data such as images and sequences, and have limitations when they are used to handle graph data. This is because graphs are non-Euclidean structures that lack a fixed size and have irregular connectivity patterns.

Graph neural networks (GNNs) have demonstrated their remarkable efficacy in handling graph data by excelling in diverse tasks, including node classification, link prediction,

and graph generation. Unlike traditional neural networks, GNNs can operate on graph-structured data and capture the dependencies between nodes in a graph [10]. Node classification is a common task in GNNs, which aims to predict the label or category of each node in a graph. It is widely used in many fields, such as predicting protein functions, identifying communities in social networks, and recommending products to users.

However, traditional GNN models, such as graph convolutional network (GCN) [11], graph sample and aggregate (GraphSAGE) [12], and graph attention network (GAT) [13], have limitations in their performance and generalization ability. For example, GCN uses a local filter to update node representations, which may lead to over-smoothing and information loss. This means that nodes that are far apart in the graph may end up with similar representations, which may weaken the model's ability to distinguish them. On the other hand, GraphSAGE and GAT use neighborhood aggregation strategies to update node representations, which may fail to capture global information and handle heterogeneous graphs. This means that the model may not be able to capture the complex relationships between different types of nodes and edges in a graph.

Motivated by these limitations researchers have proposed various extensions and improvements to GNNs. One approach is to use higher-order connectivity patterns, such as graph attention mechanisms or graph convolutional layers, to capture more nuanced information about node neighborhoods [14]. Another approach is to use meta-path-based approaches, which consider specific paths or sequences of nodes and edges to capture the connections in meaning between various types of nodes [15]. Despite these improvements, the complexity and heterogeneity of real-world graph data remain unresolved challenges that call for GNN models capable of addressing them.

To address these limitations, we propose a novel GNN ensemble model for node classification tasks, which integrates three classic GNN models: GCN, GraphSAGE, and GAT. Our proposed model combines the individual capabilities of each model and overcomes their limitations, thus improving performance and robustness. Specifically, We employ both weighted averaging and voting methods, which are two distinct approaches, to combine the predictions of individual models [16]. The weighted averaging method assigns higher weights to models with better performance, while the voting method selects the most frequent prediction among all models. We compare the performance of these two methods on different datasets and demonstrate that our proposed GNN ensemble model outperforms individual models and other classic models. Our approach provides a promising direction for improving the performance and generalization ability of GNNs, and can be utilized in diverse real-world graph applications.

The rest of this paper is structured as follows. In Section 2, we discuss the related work of our research. In Section 3, we analyze the principles and limitations of traditional GNN models. In Section 4, we introduce our proposed ensemble model and two model ensembling methods. In Section 5, we present experimental results and analysis. Finally, we conclude and summarize the paper in Section 6.

2. **Related Works.** In this section, we mainly introduce the node classification task and the mainstream variants of graph neural networks.

2.1. **Node classification task.** Node-level tasks are common applications in graph data, with node classification being the most prevalent. Previous methods mainly include iterative classification algorithms and label propagation methods, but they suffer from limitations in feature and structure information extraction and have difficulties in generalizing to real-world datasets [17, 18]. In recent years, new approaches combining random walk strategies with deep learning and graph modeling have emerged [19]. However, these

methods are still constrained by the extraction of graph topology information and the selection of mapping functions. Therefore, further research is needed to develop more effective methods to enhance the performance of node classification tasks. To address the limitations of previous methods in node classification tasks, researchers have explored various variants of GNNs.

2.2. **Variants of graph neural networks.** One popular variant is graph embedding, which maps nodes or edges in a graph to low-dimensional continuous vector space, enabling machine learning algorithms to process data organized in the form of graphs. DeepWalk [20], Node2vec [21], and Line [22] are some examples of graph embedding models. Another variant is the graph convolution network (GCN), a category of neural network that operates on data organized in the form of graphs, allowing for the modeling of relationships and dependencies between data points. GCN-LPA [23], AdaGCN [24], and N-GCN [25] are some examples of GCN models. Graph auto-encoders (GAE) are another type of GNN that can learn low-dimensional representations of graph-structured data, allowing for efficient graph compression and reconstruction while preserving important structural information. Examples of GAE models include MGAE [26] and AutoGCN [27]. Graph generative networks (GGN) are deep learning models that learn to generate graphs with desired properties, such as connectivity and node features, by leveraging graph neural networks and variational autoencoders. Graph-GAN [28] and NetGAN [29] are examples of GGN models. Graph recurrent networks (GRN) are deep learning models that can capture temporal dependencies in data organized in the form of graphs by using recurrent neural networks, rendering them appropriate for tasks such as traffic prediction and social network analysis. GGT-NN [30] and GraphRNN [31] are examples of GRN models. Lastly, the graph attention network (GAT) is a neural network architecture that uses attention mechanisms to learn node representations in a graph. Examples of GAT models include GATE [13] , RGAT [32], and GAM [33].

In conclusion, the various variants of graph neural networks offer effective instruments for extracting knowledge from data organized in the form of graphs, enabling effective representation learning, node classification, link prediction, and graph generation. These models leverage the structural information of graphs and incorporate advanced techniques such as attention mechanisms, graph convolutions, and message passing to capture complex relationships and dependencies among nodes. With their versatility and scalability, graph neural networks are poised to make significant contributions to a wide range of domains, from social network analysis to drug discovery and beyond.

3. **Overview Of Graph Neural Network Models.** A graph is a mathematical structure that consists of nodes and edges, which depict complex relationships and dependencies between entities. In a GNN, each node in the graph is assigned a feature vector that describes its properties or characteristics, while each edge is assigned a weight or feature vector that describes the relationship or interaction between the connected nodes. The ultimate objective of a GNN is to learn a function that can map the input graph to a desired output, such as classification or regression.

3.1. **Notations and preliminary.** Various commonly used graph neural network models are available, and we provide a general definition of such models in our work. Firstly, G is used to represent the input graph, where $G = (V, E)$ and V is the set of nodes while E is the set of edges. The adjacency matrix A is a non-negative matrix that encodes the pairwise relationships between nodes in the graph. The degree matrix D is a diagonal matrix that records the node degrees in the graph. The number of nodes in the graph is represented by N. The feature matrix X is an $N \times F$ matrix that describes the properties

or characteristics of each node in the graph, where F is the number of features per node. I is an identity matrix that is used to perform self-attention operations in some graph neural network models.

3.2. **Graph convolutional neural network.** GCN is a neural network architecture specifically tailored for processing data organized in the form of graphs. The key feature of GCN is that it generalizes the notion of convolution to graphs. Specifically, GCN functions by propagating information among adjacent nodes within the graph, using a graph convolution operation that combines the attributes of nodes and the structure of the graph. The propagation formula for each layer of GCN is expressed as:

$$H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}), \tag{1}$$

where $H^{(l)}$ is the node feature matrix of the l^{th} layer, A is the adjacency matrix of the graph, $\widetilde{A} = A + I$ is the adjacency matrix with added self-loops, $\widetilde{D}$ is the degree matrix of $\widetilde{A}$, $W^{(l)}$ is the weight matrix of the l^{th} layer, and σ is the activation function.

In general, we often use the formula for a two-layer GCN model, which is as follows:

$$Z = f(X, A) = softmax(\hat{A}ReLU(\hat{A}XW^{(0)})W^{(1)}), \tag{2}$$

where $\hat{A} = \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix, and the $softmax$ activation function is defined as $softmax(x_i) = \frac{1}{t}\exp(x_i)$ with $t = \sum_i \exp(x_i)$. As shown in Figure 1, the network architecture of the classic two-layer GCN is demonstrated. The current node's feature representation is updated by combining the features of individual nodes along with those of their neighboring nodes through two convolutional layers, ultimately outputting a feature matrix.

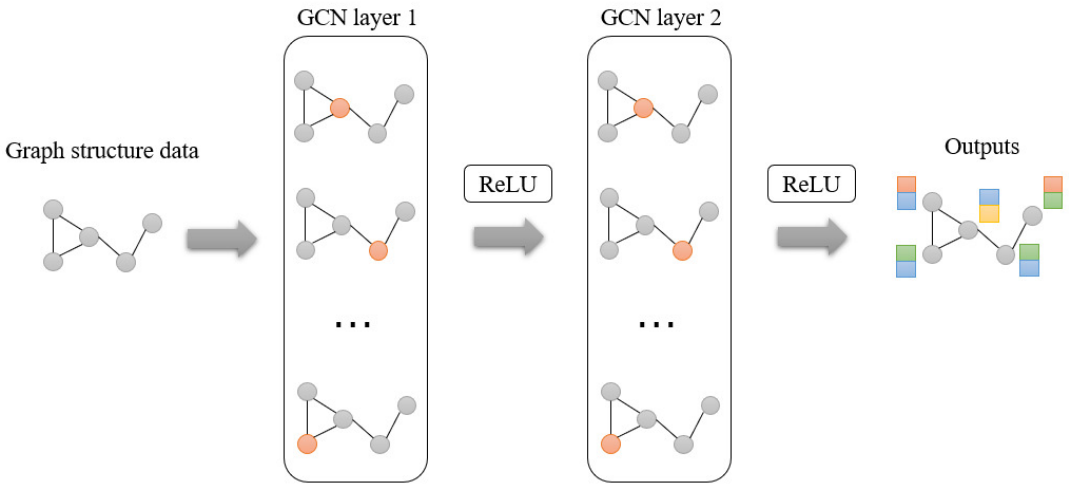

FIGURE 1. GCN two-layer architecture diagram [11]

The main advantage of GCN is its capacity for learning node representations that capture both the local and global structure of the graph. By propagating information between neighboring nodes, GCN can effectively capture the high-order connectivity patterns in the graph, leading to a powerful node representation that can be used for a variety of downstream tasks, such as node classification and link prediction.

However, GCN also has limitations. One limitation is that it is sensitive to the quality of the graph representation, which can affect its performance. Another limitation is that

GCN may suffer from over-smoothing, where node representations become too similar after multiple layers of convolution, leading to a loss of discriminative power [34]. Moreover, GCN may has difficulty in handling directed or weighted graphs, which requires modifications to the convolution operation.

3.3. **Graph sample and aggregate.** GraphSAGE is a graph representation learning algorithm. The algorithm learns node embeddings in a graph by aggregating information from the nearby neighborhood of the node. GraphSAGE operates by selectively extracting and combining features from neighboring nodes in a fixed-size receptive field around each node. The algorithm uses a multi-layer perceptron to encode these features into a low-dimensional vector representation, which can be used for downstream tasks such as node classification and link prediction. The main implementation of the algorithm can be found in Algorithm 1.

Algorithm 1 GraphSAGE embedding generation algorithm

Input: Graph G; input features x_v; depth K; weight matrices W^k; non-linearity σ; differentiable aggregator functions $AGGREGATE_K$; neighborhood function N.

Output: Vector representations Z_v.

$h_v^0 \leftarrow x_v \,, \forall v \in V$;
for $k = 1, 2, ..., K$ **do**
  for $v \in V$ **do**
    $h_{N(v)}^k \leftarrow AGGREGATE_K\left(\left\{h_u^{k-1}, \forall u \in N(v)\right\}\right)$;
    $h_v^k \leftarrow \sigma(W^k \cdot CONCAT[h_v^{k-1}, h_{N(v)}^k])$;
  end
  $h_v^k \leftarrow h_v^k / \left\|h_v^k\right\|_2$;
end
$Z_v \leftarrow h_v^K$;

For each node $v \in V$, its initial vector representation is set to the input feature x_v. The algorithm iterates K times and performs the following steps: for each node $v \in V$, aggregate the node vector representations h_u^{k-1} of its neighborhood $N(v)$, where $AGGREGATE_K$ is a differentiable aggregation function. Then update the vector representation of node v as $\sigma(W^k \cdot CONCAT[h_v^{k-1}, h_{N(v)}^k])$, where $CONCAT$ is a concatenation function. Finally, normalize the vector representation of each node $v \in V$ to have unit length. After the iterations, the vector representation of node v is set to the final representation h_v^K, and the vector representations of all nodes Z_v are outputted. The schematic diagram is shown in Figure 2.

The aggregator function we mainly use is the Mean aggregator, defined as follows:

$$h_v^k \leftarrow \sigma(W \cdot MEAN\left(\left\{h_v^{k-1}\right\} \cup \left\{h_u^{k-1}, \forall u \in N\left(v\right)\right\}\right), \tag{3}$$

Here, h_v^k represents the embedding vector of node v at layer k. $MEAN$ denotes the mean aggregation function, which takes the average of a set of vectors. W represents a learnable weight matrix. This formula indicates that the embedding vector of node v at layer k is obtained by applying mean aggregation, linear transformation, and activation function to the embedding vectors of itself and its neighboring nodes at layer $k-1$.

GraphSAGE is capable of handling large-scale graphs containing millions of nodes and edges, making it suitable for real-world applications. Additionally, GraphSAGE is highly flexible and can easily adapt to different types of graphs and tasks by adjusting the aggregation function and number of layers. However, GraphSAGE does have some limitations.

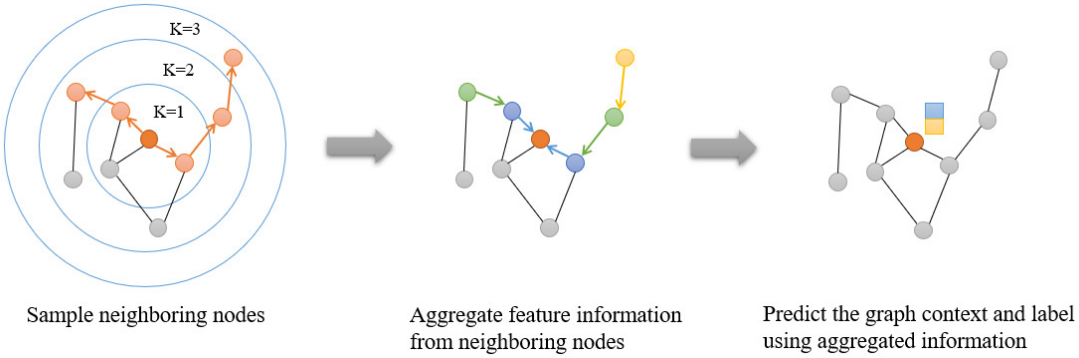

FIGURE 2. Schematic diagram of the GraphSAGE algorithm [12]

One of the main drawbacks is its reliance on a fixed-size neighborhood concept, which can potentially limit the model's capacity to capture distant relationships and global information within the graph. Another limitation is the difficulty of incorporating node features into the model, as GraphSAGE assumes all nodes have the same dimensionality [35], which may not always be the case in real-world applications.

3.4. **Graph attention network.** GAT is a GNN model that improves upon previous models by introducing an innovative attention mechanism enabling the model to selectively attend to different nodes in the graph during the aggregation process. In GAT, each node in the graph is represented by a feature vector that is updated by aggregating information from its neighbors. Unlike previous GNN models that uniformly aggregate information from all neighbors, GAT uses an attention mechanism to weight the contribution of each neighbor based on its relevance to the current node. This allows the model to give more weight to important neighbors and less weight to less relevant neighbors during the aggregation process. The attention coefficients calculation in GAT is given by the following formula.

$$e_{ij} = a(W\vec{h}_i, W\vec{h}_j), \tag{4}$$

Here, $W\vec{h}_i$ and $W\vec{h}_j$ are the transformed feature vectors for nodes i and j, respectively, obtained by multiplying the original feature vectors with a weight matrix W. a is a learnable attention mechanism.

$$\alpha_{ij} = softmax_j\left(e_{ij}\right) = \frac{\exp(e_{ij})}{\sum_{k\in N_i}\exp(e_{ik})}, \tag{5}$$

$$\alpha_{ij} = \frac{exp\left(LeakyReLU\left(\vec{a}^T\left[W\vec{h}_i \middle|\middle| W\vec{h}_j\right]\right)\right)}{\sum_{k\in N_i} exp\left(LeakyReLU\left(\vec{a}^T\left[W\vec{h}_i \middle|\middle| W\vec{h}_k\right]\right)\right)}, \tag{6}$$

The meaning of this formula is that the attention weight α_{ij} of node i is the ratio of the attention coefficient e_{ij} between node i and node j and the sum of the attention coefficients e_{ik} between node i and all of its neighboring nodes k. This is done to normalize the attention coefficients so that the sum of all attention weights of node i is equal to 1. *LeakyReLU* is an activation function used to introduce non-linearity. The symbol $||$

denotes vector concatenation, which means connecting two vectors end-to-end to form a longer vector.

According to the diagram in Figure 3. The attention mechanism in GAT is implemented using a multi-head approach, where multiple attention mechanisms are used in parallel to capture different aspects of the graph. Each attention head learns a different set of attention weights, which are combined to produce a final attention weight matrix used for aggregating neighbor information. The specific equations are as follows:

$$\vec{h}_i' = \Big\Vert_{k=1}^{K} \sigma\Big(\sum_{j \in N_i} \alpha_{ij}^k W^k \vec{h}_j\Big), \tag{7}$$

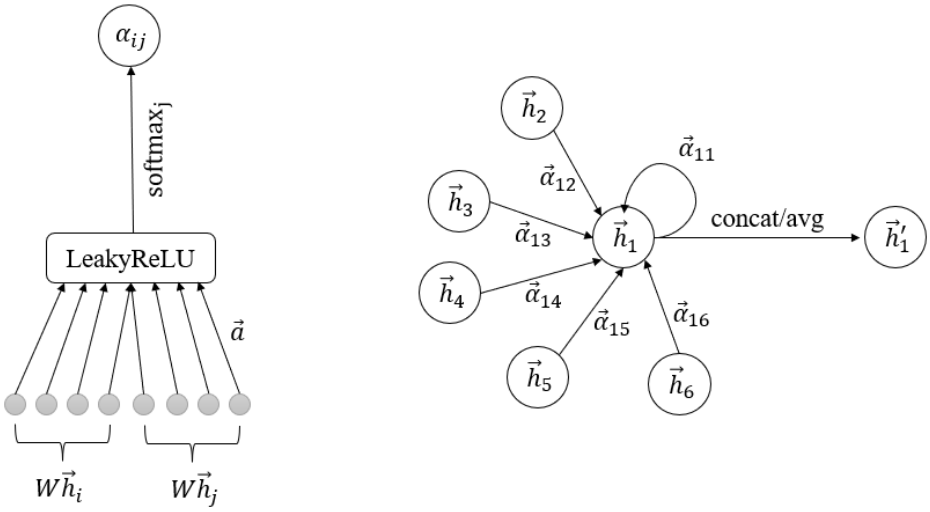

FIGURE 3. GAT multi-head attention mechanism [13]

Where $\vec{h}_i'$ represents the new feature vector of node i, which is concatenated from the outputs of multiple attention heads. K represents the number of attention heads. N_i represents the set of neighbor nodes of node i. α_{ij}^k represents the attention coefficient between node i and node j calculated by the $k-th$ attention head. W^k represents the linear transformation matrix used by the $k-th$ attention head to project the node feature vectors into the space of attention heads. $\vec{h}_j$ represents the feature vector of node j.

One of the main advantages of the GAT model is its capability to capture long-range dependencies in the graph by selectively attending to important nodes. GAT also offers a high degree of flexibility, as the number of attention heads and layers can be easily adjusted to adapt to different types of graphs and tasks. However, GAT is not without its limitations, as it requires a fixed graph structure as input, which may not always be available or may change over time. Additionally, the model's performance can be limited by the size and complexity of the input graph, as it needs to compute attention scores over all nodes and edges [36], which can become computationally expensive.

4. **The Proposed Ensemble Model.** In this module, we will be discussing a new ensemble model of graph neural networks that we have proposed. Our ensemble model is based on the parallel idea, where we consider the classic models of graph neural networks (GCN, GraphSAGE, GAT) mentioned above as independent base learning classifiers [37]. We then independently train these classifiers using the same training dataset, and finally,

we use two ensemble methods to obtain the final prediction results. In the following sections, I will offer an elaborate account of our methodologies. The specific schematic diagram is shown in Figure 4.

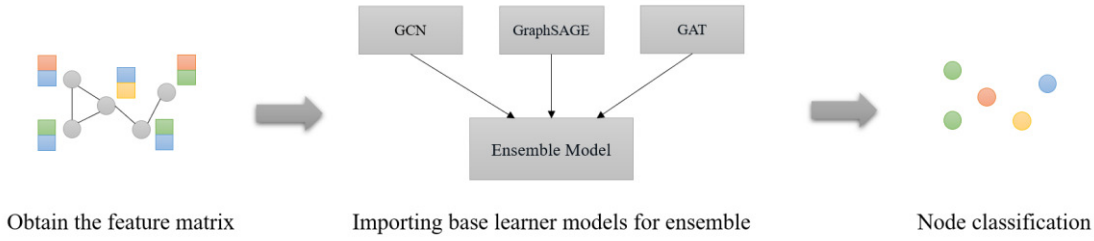

FIGURE 4. Schematic diagram of an ensemble model

4.1. **Weighted average.** The algorithmic approach we use involves taking a weighted average of the predicted results from each model to generate the final prediction. In the node classification task of graph neural networks, the weighted average method can be used to fuse the predicted results from multiple models to improve classification accuracy.

Specifically, for a given test sample, each model outputs a prediction value, which is then weighted and averaged according to pre-set weights to obtain the final prediction result. The weights can be adjusted based on model performance, data distribution, and other factors to maximize the predictive accuracy of the ensemble model. The formula for this process is as follows:

$$Logits = \frac{1}{3}\sum_{i=1}^{3} W_i \times Logits_i, \tag{8}$$

Logits refers to the output results obtained when a benchmark model predicts the test set. It is in the form of model output that has not undergone $softmax$ processing and is a two-dimensional tensor, where i corresponds to the corresponding graph neural network model. We define W_{GCN}, $W_{GraphSAGE}$, and W_{GAT} as the weight coefficients of the GCN, GraphSAGE, and GAT models used for model fusion, respectively, to control the contribution of each model in the final result.

The advantage of the weighted average method is that it can leverage the unique capabilities of diverse models to improve prediction accuracy. Additionally, it can also strengthen the model's robustness because different models may have different sensitivities to different aspects of the dataset. This improves the accuracy of the node classification task in graph neural networks.

4.2. **Voting.** Our voting strategy is as follows: for each test sample, each model predicts its class, and the prediction results are arranged into a matrix in the same order as the samples. Each column in the matrix represents a model's prediction results, and each row represents a test sample's prediction results. Then, for each row, the class with the highest frequency is counted as the final prediction result for that test sample.

Assuming we want to ensemble the above 3 graph neural network models, and each model's prediction result is $y_{i,j}$, where $i \in 1, 2, 3$ represents the $i-th$ model and $j \in 1, 2, ..., M$ represents the $j-th$ node. We use our voting strategy as the ensemble method, which means for each test node j, we choose the class with the highest frequency among all models as the ensemble model's prediction result $\hat{y}_j$. Therefore, the prediction result of the ensemble method can be represented as:

$$\hat{y}_j = argmax_{\ k} \sum_{i=1}^{3} \delta(y_{i,j} = k), \tag{9}$$

Here, $\delta(x)$ is an indicator function that equals 1 if x is true and 0 otherwise. The meaning of this formula is that for each test node j, we calculate the sum of the number of nodes predicted as class k in all models, and then select the maximum value among these sums as the ensemble model's prediction result for node j. The $argmax$ represents the class k corresponding to the maximum value after summation.

Our voting strategy integrates multiple model prediction results and selects the class with the highest frequency as the final prediction result. This can reduce the errors resulting from the inherent bias or variance of a single model, thus enhancing the precision and stability of the model. By using multiple different models for integration, the diversity of the model can be increased, mitigating the likelihood of overfitting and making the model more generalizable.

Choosing GCN, GraphSAGE, and GAT as the base learners in the graph ensemble model offers several advantages. These graph neural network models have complementary capabilities in representation and capturing graph data. GCN considers the neighborhood information of nodes, enabling it to capture the local relationships within the graph structure. It preserves node features while leveraging the graph's topology to propagate and integrate information from neighboring nodes. GraphSAGE progressively combines information from neighboring nodes at different distances through multi-layer sampling and aggregation, resulting in richer feature representations. GAT adaptsively learns the importance of each node's interactions with its neighbors and considers the relationships between different nodes simultaneously, better capturing complex relationships within the graph.

By combining these base learners using Equation (8) or Equation (9) for model ensemble, the resulting ensemble model can leverage their diverse strengths. Each model has its unique way of capturing graph features and structure. By integrating these models, the ensemble model can extract a wider and more comprehensive range of information from the input graph, enhancing its understanding of the graph's complexity. Additionally, combining multiple models helps reduce biases and limitations of individual models, resulting in more reliable and robust predictions.

4.3. **Optimization.** The defined loss function encompasses the cross-entropy error across all labels.

$$\mathcal{L} = -\sum_{i \in \mathcal{V}_l} \sum_{f=1}^{F} Y_{if} ln Z_{if}, \tag{10}$$

Where $\mathcal{V}_l$ is the set of indices of the labeled vertices, F is the dimensionality of the output features, equivalent to the number of classes, and $Y \in R^{|\mathcal{V}_l| \times F}$ is a label indicator matrix.

5. Experiment.

5.1. **Datasets.** We employed the widely-used citation network datasets, Cora, Citeseer, and Pubmed, as our primary data sources [38, 39] for conducting node classification tasks using graph neural networks. To facilitate data loading and preprocessing, we leveraged the Planetoid class provided by PyTorch Geometric (PyG) library. The datasets were meticulously organized and stored locally for seamless access. For the purpose of model

training and evaluation, we followed the default configuration of the Planetoid class. Specifically, each class category was allocated 20 samples for the training set, 500 samples for the validation set, and 1000 samples for the test set. Notably, the assignment of nodes to each set was carried out via random sampling, ensuring a representative distribution. The Table 1 shows the specific information for each dataset:

Table 1. Datasets overview in our experiments

Datasets	*Nodes*	*Edges*	*Features*	*Classes*
Cora	2,708	10,556	1,433	7
Citeseer	3,327	9,104	3,703	6
Pubmed	19,717	88,648	500	3

5.2. **Baselines.** In addition to the three classic graph neural network models mentioned above, we also employed three more advanced baseline models, namely SGC, DGI and HGCN.

DGI [40](Deep Graph Infomax) is a graph neural network algorithm that learns node representations by maximizing the mutual information between local patch representations and global graph representations. It has achieved excellent performance on several benchmark graph classification tasks.

SGC [41] (Simplifying Graph Convolutional Networks) is a model that simplifies the complexity of graph convolutional networks. It stands out for its simplicity, offering a streamlined implementation while maintaining impressive performance.

HGCN [42] (Hyperbolic Graph Convolutional Neural Networks) is a model that leverages hyperbolic geometry for graph convolutional operations. It excels in capturing hierarchical structures and long-range dependencies in complex networks, leading to enhanced representation learning capabilities.

5.3. **Experiment preparation.** We employed two layers of GCN, GraphSAGE, and GAT models as independent base learners for our classification task. To avoid overfitting, we applied Dropout technique during training. We set the number of attention heads K in Equation (7) to 8. The hidden-channels for GCN and GraphSAGE were set to 16, while that for GAT was set to 8. We optimized the models using the Adam optimizer with a learning rate of 0.01 and $L2$ regularization strength of 5e-4. To facilitate better analysis and comparison of experimental data, we have named the ensemble model based on Equation (8) as EGNN-W and Equation (9) as EGNN-V. We conducted separate experiments for both ensemble models. Based on the actual performance of the model classification node tasks, for the Cora dataset, we set W_{GAT} to 1.5, $W_{GraphSAGE}$ and W_{GCN} to 1. For the Citeseer dataset, we set W_{GAT} to 1.4, $W_{GraphSAGE}$ to 1.3, and W_{GCN} to 1. For the Pubmed dataset, we set W_{GCN} to 1.3, $W_{GraphSAGE}$ and W_{GAT} to 1.

5.4. **Experimental evaluation and results.** In our evaluation, we utilize two key performance metrics, Accuracy and Macro-F1 score, to assess the effectiveness of our models. These metrics provide us with a quantitative measure of how well our models are able to classify and predict various outcomes.

Accuracy is a simple and commonly used metric that measures the percentage of correctly classified instances over the total number of instances. It can be calculated using the formula:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}, \tag{11}$$

Where TP denotes the count of true positives, TN denotes the count of true negatives, FP denotes the count of false positives, and FN denotes the count of false negatives.

In addition to Accuracy, we also use Macro-F1 score as a performance metric. The Macro-F1 score relies on two additional metrics, namely precision and recall. Precision quantifies the ratio of true positive predictions to all positive predictions, while recall quantifies the ratio of true positive predictions to all actual positive instances. The formula for precision and recall are:

$$Precision\ P = \frac{TP}{TP + FP}, \tag{12}$$

$$Recall\ R = \frac{TP}{TP + FN}, \tag{13}$$

Macro-F1 can then be calculated using the following formula:

$$Macro - F1 = \frac{1}{N} \sum_{i=1}^{N} \frac{2 \times P_i \times R_i}{P_i + R_i}, \tag{14}$$

Where N represents the total number of categories, P_i represents the precision of the $i - th$ category, and R_i represents the recall of the $i - th$ category.

To ensure the reliability of our experiments, we conducted a comparison of all models within the same environment. Each model underwent 200 epochs and was tested over 10 runs to evaluate its ability to classify nodes across various datasets. The specific data statistics are presented in the following Table 2. To demonstrate the superiority of our proposed ensemble models, we compare the performance of multiple models using bar charts, as shown in Figure 5.

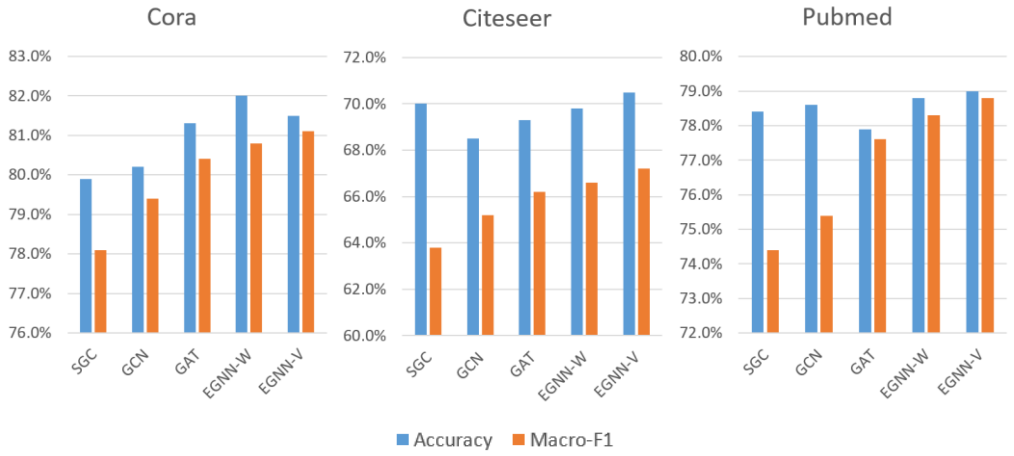

FIGURE 5. Performance comparison of different datasets

5.5. **Ensemble method analysis.** Based on the experimental results, we observed that EGNN-V generally outperforms EGNN-W across various datasets. We analyzed several potential reasons for this trend.

Firstly, voting ensemble models are typically composed of multiple independent models, each with its own learning characteristics and preferences. This diversity facilitates the ensemble model in capturing different aspects and patterns of the data more effectively.

TABLE 2. Comparison of Accuracy and F1-score of different models on different datasets

	Accuracy		
Models	*Cora*	*Citeseer*	*Pubmed*
DGI	80.2±0.7	68.5±0.6	76.5±0.4
SGC	79.9±0.5	70.0±0.4	78.4±0.5
HGCN	79.5±0.7	68.3±0.4	78.1±0.3
GCN	80.2±0.8	68.5±0.5	78.6±0.3
GAT	81.3±0.6	69.3±0.6	77.9±0.3
GraphSAGE	80.4±0.4	69.2±0.7	77.9±0.4
EGNN-W	**82.0±0.4**	69.8±0.3	78.8±0.3
EGNN-V	81.5±0.3	**70.5±0.3**	**79.0±0.2**
	Macro-F1		
Models	*Cora*	*Citeseer*	*Pubmed*
DGI	78.5±0.3	62.4±0.4	72.2±0.4
SGC	78.1±0.3	63.8±0.5	74.4±0.3
HGCN	78.4±0.3	63.3±0.3	75.3±0.2
GCN	79.4±0.4	65.2±0.3	75.4±0.2
GAT	80.4±0.3	66.2±0.3	77.6±0.3
GraphSAGE	79.6±0.3	64.5±0.3	76.2±0.3
EGNN-W	80.8±0.2	66.6±0.2	78.3±0.3
EGNN-V	**81.1±0.2**	**67.2±0.3**	**78.8±0.2**

In contrast, weighted averaging models may not fully account for the differences between individual models during weight allocation, thus potentially failing to harness the diversity among the models efficiently.

Secondly, in a voting ensemble, when different models generate disparate predictions, the final result is determined through voting, which can offset the erroneous predictions made by individual models. This error compensation effect helps reduce the overall model's bias and variance, thereby enhancing its stability and performance.

Lastly, voting ensemble models generally exhibit better robustness to noise and outliers. As the voting result is based on the consensus of multiple models, errors made by individual models are often counterbalanced by the correct predictions of others. In contrast, weighted averaging models are more sensitive to the errors of individual models since incorrect weight allocation can lead to a decline in the overall model's performance.

5.6. **Base learners analysis.** We observed that in our experiments, the performance of the ensemble model gradually improved as we added base learners step by step. Taking the Cora dataset as an example, where 1 represents GCN, 2 represents GCN+GraphSAGE, and 3 represents GCN+GraphSAGE+GAT, we employed both weighted averaging and voting as independent ensemble methods, and the performance of the ensemble models is shown in Figure 6. However, we noticed that when we added more base learners, the performance didn't show significant improvements and it consumed more time. We analyze the following reasons for this:

As we progressively added GCN, GraphSAGE and GAT, the performance of the ensemble model improved gradually. This is because these models are effective in capturing the relationships between nodes and the structural information of the graph when dealing with graph-structured data. The introduction of each model increases the expressive power, leading to improved performance. When we added more base learners, it's possible

that the model's expressive power had already reached a saturation point, and further additions didn't contribute significantly to performance improvement. In such cases, adding more learners may not result in substantial changes in performance. If the added base learners are more complex, it can lead to overfitting on the training data. Overfitting causes the model's performance to decline on unseen data, which could explain why the performance didn't improve significantly. Adding more base learners increases the complexity and computational requirements of the model, resulting in longer training and inference times. This also explains why it consumed more time when more learners were added.

In practical applications, it's important to strike a balance between model performance and time consumption, and select the ensemble method and model configuration that best suits the task requirements. It may require further experimentation, analysis, and fine-tuning, including model selection, hyperparameter tuning, to find the optimal trade-off between performance and time efficiency.

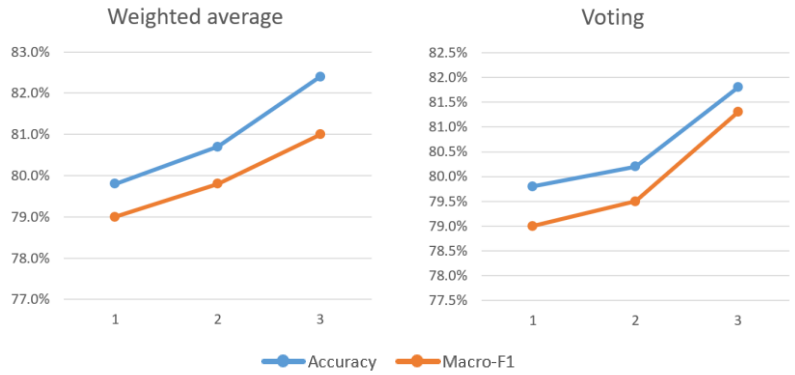

FIGURE 6. Improvement of ensemble model with added base learners

6. **Conclusion.** Our study demonstrates the effectiveness of ensemble models in improving the performance of classical graph neural network models. Specifically, our ensemble model combines the results of three models, namely GAT, GraphSAGE, and GCN, and achieves better performance than any single model on the Cora, Citeseer, and Pubmed datasets. We also observe that the ensemble model is more robust to noise and outliers, which is crucial in real-world applications where data quality is often a concern. The success of our ensemble model can be attributed to several factors. Firstly, the combination of multiple models can reduce the impact of individual model weaknesses and leverage their strengths. Secondly, the ensemble model can efficiently capture the diversity of the data and produce more accurate predictions by combining the predictions generated by multiple models. Thirdly, the ensemble model can reduce overfitting by balancing the predictions generated by multiple models and removing the bias of any single model. Overall, our study suggests that ensemble models can be a powerful tool for improving the performance of graph neural network models. Future research can explore the optimal way to combine different models and investigate the generalizability of ensemble models across different graph datasets and tasks.

Acknowledgment. This work is supported by the Natural Science Foundation of Fujian Province under Grant 2023J011430 and the Natural Science Foundation of Xiamen under Grant 3502Z20227067. The authors would like to express their gratitude for the valuable comments and suggestions provided by the reviewers, which have significantly enhanced

the quality of the presentation. Ke-Ao Lin proposed the methodology, conducted experiments and wrote the manuscript; Xiao-Zhu Xie and Wei Weng revised the manuscript and made constructive comments; Yong Chen participated in the discussion.

REFERENCES

[1] N. Liu, Q. Feng, X. Hu, "Interpretability in Graph Neural Networks," *Graph Neural Networks: Foundations, Frontiers, and Applications*, pp. 121-147, 2022.

[2] F. Zhang, T.-Y. Wu, Y. Wang, R. Xiong, G. Ding, P. Mei, and L. Liu, "Application of quantum genetic optimization of LVQ neural network in smart city traffic network prediction," *IEEE Access*, vol. 8, pp. 104555-104564, 2020.

[3] K. Wang, Z. Chen, X. Dang, X. Fan, X. Han, C.-M. Chen, W. Ding, S.-M. Yiu, and J. Weng, "Uncovering Hidden Vulnerabilities in Convolutional Neural Networks through Graph-based Adversarial Robustness Evaluation," *Pattern Recognition*, pp. 109745, 2023.

[4] K. Wang, Z. Chen, M. Zhu, S.-M. Yiu, C.-M. Chen, M.M. Hassan, S. Izzo, and G. Fortino, "Statistics-Physics-Based Interpretation of the Classification Reliability of Convolutional Neural Networks in Industrial Automation Domain," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 2165-2172, 2022.

[5] L. Lin, Z. Xu, C.-M. Chen, K. Wang, M.R. Hassan, M.G.R. Alam, M.M. Hassan, and G. Fortino, "Understanding the impact on convolutional neural networks with different model scales in AIoT domain," *Journal of Parallel and Distributed Computing*, vol. 170, pp. 1-12, 2022.

[6] D. Ghimire, D. Kil, S. Kim, "A survey on efficient convolutional neural networks and hardware acceleration," *Electronics*, vol. 11, no. 6, pp. 945, 2022.

[7] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999-7019, 2022

[8] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235-1270, 2019.

[9] H. Hewamalage, C. Bergmeir, K. Bandara, "Recurrent neural networks for time series forecasting: Current status and future directions," *International Journal of Forecasting*, vol. 37, no. 1, pp. 388-427, 2021.

[10] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S.Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4-24, 2020.

[11] T.N. Kipf, M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv*, 2016. [Online]. Available: https://arxiv.org/abs/1609.02907

[12] W. Hamilton, Z. Ying, J. Leskovec, "Inductive representation learning on large graphs," *Advances in Neural Information Processing Systems*, pp. 30, 2017.

[13] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv*, 2017. [Online]. Available: https://arxiv.org/abs/1710.10903

[14] A.G. Carranza, R.A. Rossi, A. Rao, and E. Koh, "Higher-order clustering in complex heterogeneous networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. SIGKDD, 2020, pp. 25-35.

[15] P. Pham, P. Do, "W-MetaPath2Vec: The topic-driven meta-path-based model for large-scaled content-based heterogeneous information network representation learning," *Expert Systems with Applications*, vol. 123, pp. 328-344, 2019.

[16] M.A. Ganaie, M. Hu, A.K. Malik, M. Tanveer, and P.N. Suganthan, "Ensemble deep learning: A review," *Engineering Applications of Artificial Intelligence*, vol. 115, pp. 105151, 2022.

[17] Z. Huang, Y. Tang, Y. Chen, "A graph neural network-based node classification model on class-imbalanced graph data," *Knowledge-Based Systems*, vol. 244, pp. 108538, 2022.

[18] S.K. Maurya, X. Liu, T. Murata, "Simplifying approach to node classification in graph neural networks," *Journal of Computational Science*, vol. 62, pp. 101695, 2022.

[19] M. Zou, Z. Gan, R. Cao, C. Guan, and S. Leng, "Similarity-navigated graph neural networks for node classification," *Information Sciences*, vol. 633, pp. 41-69, 2023.

[20] B. Perozzi, R. Al-Rfou, S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. SIGKDD, 2014, pp. 701-710.

[21] A. Grover, J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. SIGKDD, 2016, pp. 855-864.
[22] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*. WWW, 2015, pp. 1067-1077.
[23] H. Wang, J. Leskovec, "Unifying graph convolutional neural networks and label propagation," *arXiv*, 2020. [Online]. Available: https://arxiv.org/abs/2002.06755
[24] K. Sun, Z. Zhu, Z. Lin, "Adagcn: Adaboosting graph convolutional networks into deep models," *arXiv*, 2019. [Online]. Available: https://arxiv.org/abs/1908.05081
[25] S. Abu-El-Haija, A. Kapoor, B. Perozzi, and J. Lee, "N-gcn: Multi-scale graph convolution for semi-supervised node classification," in *Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 841-851.
[26] S. Kou, W. Xia, X. Zhang, Q. Gao, and X. Gao, "Self-supervised graph convolutional clustering by preserving latent distribution," *Neurocomputing*, vol. 437, pp. 218-226, 2021.
[27] J.X. Luo, Y.J. Du, "Detecting community structure and structural hole spanner simultaneously by using graph convolutional network based Auto-Encoder," *Neurocomputing*, vol. 410, pp. 138-150, 2020.
[28] Z. Gharaee, S. Kowshik, O. Stromann, and M. Felsberg, "Graph representation learning for road type classification," *Pattern Recognition*, vol. 120, pp. 108174, 2021.
[29] S. Zhang, W. Ni, N. Fu, "Differentially private graph publishing with degree distribution preservation," *Computers & Security*, vol. 106, pp. 102285, 2021.
[30] F.R. Bach, M.I. Jordan, "Learning graphical models for stationary time series," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2189-2199, 2004.
[31] J. Liu, A. Kumar, J. Ba, J. Kiros, and K. Swersky, "Graph normalizing flows," *Advances in Neural Information Processing Systems*, pp. 32, 2019.
[32] Y. Zhao, H. Zhou, A. Zhang, R. Xie, Q. Li, and F. Zhuang, "Connecting embeddings based on multiplex relational graph attention networks for knowledge graph entity typing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 5, pp. 4608-4620, 2022.
[33] J. Wu, S. Pan, X. Zhu, C. Zhang, and S.Y. Philip, "Multiple structure-view learning for graph classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 3236-3251, 2017.
[34] Y. Yan, M. Hashemi, K. Swersky, Y. Yang, and D. Koutra, "Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks," in *2022 IEEE International Conference on Data Mining (ICDM)*. ICDM, 2022, pp. 1287-1292.
[35] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," *Advances in Neural Information Processing Systems*, pp. 32, 2019.
[36] S. Brody, U. Alon, E. Yahav, "How attentive are graph attention networks?," *arXiv*, 2021. [Online]. Available: https://arxiv.org/abs/2105.14491
[37] K. Zhou, Y. Yang, Y. Qiao, and T. Xiang, "Domain adaptive ensemble learning," *IEEE Transactions on Image Processing*, vol. 30, pp. 8008-8018, 2021.
[38] Z. Yang, W. Cohen, R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *International Conference on Machine Learning*. PMLR, 2016, pp. 40-48.
[39] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93-93, 2008.
[40] P. Veličković, W. Fedus, W.L. Hamilton, P. Liò, Y. Bengio, and R.D. Hjelm, "Deep graph infomax," *arXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1809.10341
[41] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6861-6871.
[42] I. Chami, Z. Ying, C. Ré, and J. Leskovec, "Hyperbolic graph convolutional neural networks," *Advances in Neural Information Processing Systems*, pp. 32, 2019.