EFFICIENT ENSEMBLES OF GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph Neural Networks (**GNNs**) have enabled the power of deep learning to be applied to inputs beyond the Euclidean domain, with applications ranging from social networks and product recommendation engines to the life sciences. GNNs, like other classes of machine learning models, benefit from ensemble learning, wherein multiple models are combined to provide higher accuracy and robustness than single models. However, ensembles suffer from significantly higher inference processing and storage requirements, limiting their use in practical applications. In this work, we leverage the unique characteristics of GNNs to overcome these overheads, creating efficient ensemble GNNs that are faster than even single models at inference time. We observe that during message passing, nodes that are incorrectly classified (error nodes) also end up adversely affecting the representations of other nodes in their neighborhood. This error propagation also makes GNNs more difficult to approximate (*e.g.*, through pruning) for efficient inference. We propose a technique to create ensembles of diverse models, and further propose Error Node Isolation (ENI), which prevents error nodes from sending messages to (and thereby influencing) other nodes. In addition to improving accuracy, ENI also leads to a significant reduction in the memory footprint and the number of arithmetic operations required to evaluate the computational graphs of all neighbors of error nodes. Remarkably, these savings outweigh even the overheads of using multiple models in the ensemble. A second key benefit of ENI is that it enhances the resilience of GNNs to approximations. Consequently, we propose Edge Pruning and Network Pruning techniques that target both the input graph and the neural networks used to process the graph. Our experiments on GNNs for transductive and inductive node classification demonstrate that ensembles with ENI are simultaneously more accurate (by up to 4.6% and 3.8%) and faster (by up to 2.8× and 5.7×) when compared to the best-performing single models and ensembles without ENI, respectively. In addition, GNN ensembles with ENI are consistently more accurate than single models and ensembles without ENI when subject to pruning, leading to additional speedups of up to 5× with no loss in accuracy.

1 INTRODUCTION

Graph Neural Networks (GNNs) such as GCN (Kipf & Welling, 2017), GraphSage (Hamilton et al., 2017) and JKNet (Xu et al., 2018) have revolutionized the field of graph-based learning with their ability to process and understand a diverse range of graphs (social media networks, molecular structures, road maps, etc.) using both topological and feature information. Each node in the graph builds its representation based on its own features as well as the features of its neighbors through message passing between neighboring nodes. Nodes receive messages from all other nodes in their L-hop neighborhood (where L is the number of layers in the GNN), and update their representations using an aggregation of these messages. Therefore, the time (and memory) complexity of message passing is exponential in the number of GNN layers, making real-world large-scale GNNs highly compute-intensive. Ensemble learning is a popular technique to boost the generalization performance, training stability and robustness of machine learning applications, and GNNs have also been shown to benefit greatly from the use of ensembles (Kosasih et al., 2021). However, ensembles suffer from significantly greater inference latency and storage requirements (both of which increase linearly with the number of models in the ensemble) than single models, and this becomes especially problematic for the already compute-intensive GNNs. In this work, we use the unique characteristics of GNNs to overcome the challenges of ensemble learning, enabling GNNs to reap its benefits without any drawbacks at inference time.

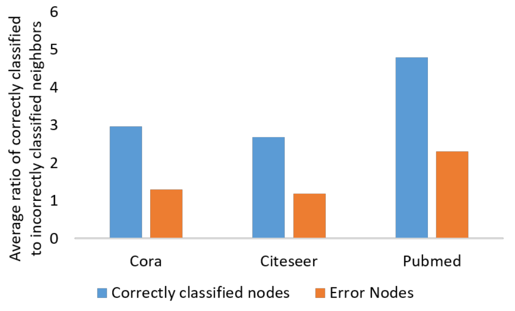

Figure 1: **Error nodes have a detrimental effect on the representations of nodes in their neighborhood.** Compared to correctly classified nodes, error nodes receive messages from a significantly larger proportion of other error nodes, which adversely affects their classification.

We focus on GNNs used for node classification, a popular problem with high practical significance. Due to the use of message passing in GNNs, the individual samples are not independent of each other, and perturbations in the representation of one node affect the representations of all other nodes in its neighborhood. We observe that messages from nodes that are incorrectly classified (which we call *error nodes* for brevity) have an adverse effect on the representations of all neighboring nodes (as seen in Fig. 1). Indeed, the Single Node Attack (Finkelshtein et al., 2020) leverages this phenomenon, creating adversarial perturbations in a single node that cause a large drop in node classification accuracy across the entire graph. Error nodes interfere with the functionality of the message passing pipeline, leading to propagation of misclassifications (unlike traditional DNNs, where the misclassification of one sample does not affect the classification of other samples). Therefore, *Error Node Isolation (ENI)* is vital for preventing error nodes from sending messages to the other nodes. Some error nodes are labeled (part of training/validation set), and they can be identified and isolated during training. However, many error nodes are also unlabeled, especially since most real-world graphs are sparsely labeled. Fortunately, ensembles of diverse models can also be used to detect unlabeled error nodes, based on divergence among the predictions of constituent models in the ensemble. The use of ENI provides two major benefits: (1) ENI improves information flow in the model by eliminating harmful and/or noisy messages from error nodes, leading to better node representations and hence, better generalization performance. (2) ENI improves inference efficiency, since messages from error nodes do not need to be computed and aggregated. Due to the exponential complexity of message passing, the large decrease in inference operations, dynamic memory usage and runtime as a result of ENI actually outweighs the increase due to the use of multiple models in the ensemble.

We also find that single-model GNNs are difficult to approximate (through techniques such as pruning) for efficient inference, due to the propagation of misclassifications in the graph. Nodes that become error nodes (as a result of approximations) adversely affect all nodes in their neighborhoods, resulting in new error nodes. These new error nodes transitively cause further error nodes in their neighborhoods (and so on), leading to a large drop in accuracy when approximations are applied aggressively for highly efficient inference. The use of ENI drastically reduces this effect, and hence, enhances the resilience of GNNs to approximations. In this work, we propose Graph neural network Ensembles with ENI (*GEENI*), and introduce an ensemble creation technique that produces ensembles of diverse models that are capable of identifying and isolating error nodes. We also propose two pruning techniques that take advantage of the increased resilience of GEENI to approximations: (1) *Edge pruning* finds the largest set of unimportant edges in the input graph that can be pruned for a given accuracy constraint. Complementary to ENI, edge pruning removes unimportant edges between correctly classified nodes. Since predictions from multiple models in the ensemble are used to produce the final result, the graph can be pruned significantly more than in single models, with little to no effect on final accuracy. Due to the increased sparsity of the graph and the exponential complexity of message passing, inference using models produced by GEENI is significantly faster than a single model processing a larger, denser graph. (2) *Network pruning* performs structured pruning of neural networks processing the graph, enabling further gains in efficiency. We summarize our main contributions as follows:

- We introduce a framework for creating Graph neural network ensembles with ENI (*GEENI*) that are *both faster and more accurate* than traditional single models.
- We propose an Ensemble Creation technique to create ensembles of diverse models that can effectively perform ENI.
- We propose pruning techniques for both the input graph and the underlying neural networks that take advantage of the increased resilience of GEENIs for efficient and accurate inference.
- Across four graph datasets composed of transductive (no new nodes are added to the graph at testing time; the task is to predict unlabeled nodes in the original graph) and inductive (new, unseen nodes are added to the graph at testing time) node classification tasks, GEENI produces models that are consistently more accurate and consistently faster than the best-performing single models and ensembles without ENI. In addition, we demonstrate that GEENI models are consistently more accurate than the best-performing single models with the same inference speed when aggressively pruned.

2 Related Work

Ensemble Learning. Ensembles have been widely adopted to improve the accuracy, stability and robustness of machine learning systems (Hansen & Salamon, 1990; Dietterich, 2000; Strauss et al., 2017). However, these ensembles add significant computational and memory overheads, and techniques have been proposed (Wen et al., 2020; Sen et al., 2020; Ruiz & Verbeek, 2021) to address this challenge. Complementary to these, we use the unique properties of GNNs to further reduce the overheads, enabling ensemble GNNs to be faster than even single models at inference time. Another line of research focuses on improving ensemble diversity so that different models in the ensemble do not make the same mistakes (Zhang et al., 2020; Kariyappa et al., 2021; Pang et al., 2019). However, we find that compared to other machine learning applications, creating diverse GNNs is significantly more challenging, since it has been shown that GNN predictions resemble label propagation (Yang et al., 2020; Wang & Leskovec, 2020), where models trained on the same input graph and training dataset are likely to make the same predictions.

Improving the efficiency of GNNs. The vast majority of prior work on improving the efficiency of GNNs is focused on fast and scalable training of large graphs. The most prominent technique is graph sampling (Chen et al., 2018; Huang et al., 2018; Hamilton et al., 2017), where only a small subset of neighbors is considered for each node during training, thereby reducing the size of the computation graph for each node and enabling efficient mini-batch training. Relatively less attention has been paid to improving the inference efficiency of GNNs. Jia et al. (2020) identify and eliminate redundant computations during message passing and aggregation. Xu et al. (2020) restrict message passing to input-dependent subgraphs. Yan et al. (2020) learn smaller GNNs that can match the performance of large GNNs through knowledge distillation. Tailor et al. (2021) enable the use of 8-bit integer arithmetic for inference through quantization. Li et al. (2021) use neural architecture search to find efficient networks. Chen et al. (2021) use Lottery Ticket Hypothesis to find sparse graphs and models for efficient inference. Our work is complementary to these efforts, and GEENI enhances the resilience of GNNs to approximations, thereby making them more effective. We also demonstrate that GEENI can produce more efficient models for a given accuracy constraint than Chen et al. (2021), with minimal post-training overheads.

3 Ensemble Creation and Inference

Our framework is illustrated in Fig. 2. Ensembles of diverse models have been shown to be significantly more accurate, stable and robust than single models. However, creating sufficiently diverse GNNs is challenging, since it has been previously observed that GNN predictions resemble label propagation (Wang & Leskovec, 2020). Consequently, the predictions of GNNs mainly depend on the topology of the input graph and the labeled nodes, and hence, models trained on the same input graph using the same training set are highly likely to make the same predictions (and mistakes). We use a combination of multiple regularizers during training to overcome this challenge. DropNode (Feng et al., 2020) randomly drops nodes from the input graph and the training set, and DropEdge (Rong et al., 2020) randomly drops edges from the training graph during each training iteration.

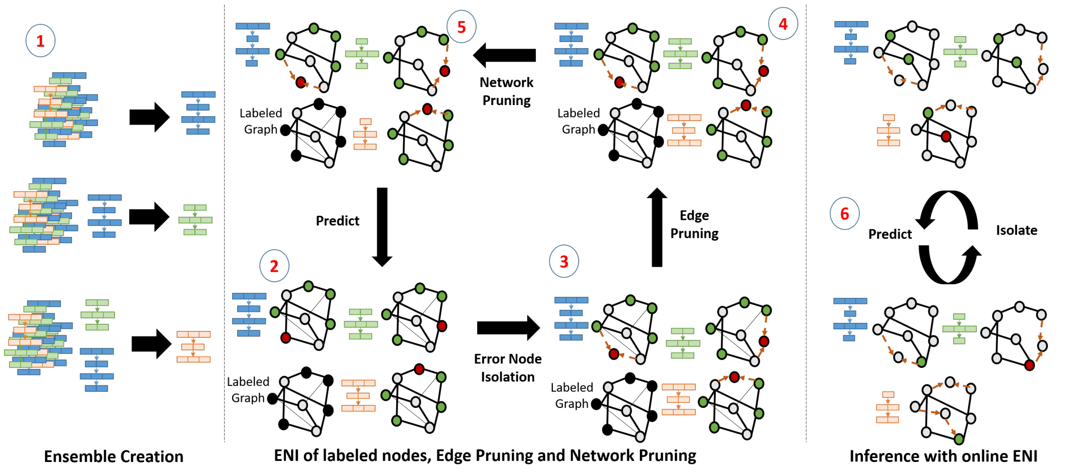

Figure 2: **Illustrations of the different techniques proposed in this work.** (1) In Ensemble Creation, the highest accuracy model is chosen as the base model from a universe of possible models. The remaining models are chosen to balance accuracy vs. diversity with the models already in the ensemble (2) Each model in the ensemble is asked to predict all the labeled (black) nodes (3) Nodes whose predictions do not match the labels (red) are isolated (not allowed to send messages to other nodes; can still receive messages) (4) Edge Pruning prunes the least important edges between correctly classified nodes (5) Network Pruning prunes the least important parameters in each model (6) At inference time, each model predicts the unlabeled (gray) nodes. If a model's prediction does not match the majority prediction of the ensemble (red), the node is isolated in the model.

Sampling restricts message passing to a small subset of neighbors for each node. The combination of DropNode, DropEdge and sampling significantly alters the graph topology and the training set in each training iteration, leading to models that are more diverse. To further increase model diversity, we also use regularizers on the neural network that processes the input graph. Dropout randomly drops weights from the neural network, and Structured Dropout (Fan et al., 2020) randomly drops large, structured groups of weights (such as neurons or even entire GNN Layers).

The Ensemble Creation procedure is described in Alg. 1. First, the base model in the ensemble is created using the best known hyperparameter values and network architecture for the given task. For instance, on Citeseer (full-supervised), the current state-of-the-art model (4-layer IncepGCN with the appropriate hyperparameters) is selected as the base model. In order to add more models to the ensemble, multiple models of a given network architecture type are trained using different regularizer rates and hyperparameter values. Then, each model is scored based the extent to which adding the model to the ensemble improves the ensemble quality (accuracy + diversity) using the $compute\_ensemble\_quality\_score$ function, and the model with the highest score is added to the ensemble. The $compute\_ensemble\_quality\_score$ function improves diversity of the ensemble (by promoting models that make dissimilar predicitons for nodes misclassified nodes by the ensemble), while also ensuing that the models in the ensemble are highly accurate (by promoting models that make correct predicitons for nodes correctly classified nodes by the ensemble). This, in turn, leads to high quality ensembles composed of accurate and diverse models, and equipped with the ability to effectively perform ENI.

At inference time, we fuse the different models in GEENI into a single multi-branch model for efficient parallel execution on the underlying hardware. This implementation is enabled by two unique characteristics of GNNs: (1) The models used in GNNs tend to be very small (around 5MB) compared to the input graphs (few GBs) in most cases (with some exceptions, such as GNNs used in molecular prediction, where the input graphs are also small). The node features (most memory-consuming part of a graph) are a function of the dataset, and hence, only one copy of the node features needs to be stored. Each model in the ensemble maintains its own individual copy of the adjacency matrix with the appropriate nodes isolated, which is stored in sparse format to minimize memory usage. Hence, the static memory overhead of using additional models in the ensemble is small (Appendix A). (2) The large decrease in inference operations and dynamic memory usage as a

Algorithm 1: Ensemble Creation

```
Function $compute\_ensemble\_quality\_score$ (model to evaluate $M$, validation set $V$,
  ensemble state $E$) :
    $ensemble\_quality\_score$ = 0
    for each node $N$ in $V$ do
        $model\_prediction$ = classify_node($M$,$N$)
        $ensemble\_prediction$ = classify_node($E$,$N$)
        if ($ensemble\_prediction$ is wrong and $ensemble\_prediction \neq model\_prediction$) or
         ($ensemble\_prediction$ is correct and $ensemble\_prediction = model\_prediction$)
         then
            $ensemble\_quality\_score = ensemble\_quality\_score$ + 1
    return $ensemble\_quality\_score$

$N$: Number of models in the ensemble
$K$: Number of models of each network type considered for addition to the ensemble
$Ensemble$ = {}
$Base\_Model$ = Best Known Model (Dataset)
$Ensemble$ = $Ensemble \cup Base\_Model$
for $i$ = 1 to N-1 do
    $models$ = Train $K$ different models of network architecture $i$ with varying amounts of
      regularization and different hyperparameters
    $max\_ensemble\_quality\_score$ = 0
    for $j$ = 1 to $K$ do
        $model\_score$ = $compute\_ensemble\_quality\_score$($models[j]$, V, Ensemble)
        if $model\_score > max\_ensemble\_quality\_score$ then
            $max\_ensemble\_quality\_score$ = $model\_score$
            $best\_model$ = $models[j]$
    $Ensemble$ = $Ensemble \cup best\_model$
return Ensemble
```

result of ENI outweighs the increase due to the use of multiple models in the ensemble. Therefore, dynamic memory overheads of the ensemble are negligible (and in most cases, less than single models).

4 Error Node Isolation (ENI)

In order to prevent propagation of misclassifications in the graph, we propose ENI to identify and isolate the error nodes, preventing them from sending messages to other nodes. Some notable reasons for the existence of error nodes after training include the presence of outliers in the input graph, the use of sampling during training, the use of approximations to improve training and inference efficiency, adversarial attacks, and the inherently approximate nature of neural networks. We define isolating a node as pruning all outgoing edges (and hence, suppressing all outgoing messages) from it. Isolated nodes retain their incoming edges, so that they can be predicted. When new, unseen neighbors are added to the error nodes, they are inspected again, and reintroduced into the graph if found to be correctly classified. We note that in order to equip the ensemble with the ability to effectively perform ENI, we need to ensure that the models in the ensemble have sufficient diversity, so that they do not make the same mistakes. We perform ENI in two stages:

Post-training (pre-inference) ENI of labeled nodes. Some error nodes are part of the training/validation set, and these can be easily identified and isolated. All the labeled nodes are predicted (classified) after training, and nodes whose predictions do not match the labels are identified as error nodes and isolated. This process is repeated after any approximation is applied to the input graph or the neural networks processing the graph, since approximations can create additional error nodes.

Online ENI of unlabeled nodes during inference. The vast majority of error nodes in the graph are unlabeled, especially since real-world graphs are sparsely labeled. We use an ensemble of diverse models to predict the unlabeled error nodes. All models in the ensemble predict a certain node. If the prediction of a model in the ensemble does not match the majority prediction, then the node is flagged as an error node, and isolated in the graph stored by the model. If all models provide different predictions (no consensus in predictions), then the node is isolated in the graphs stored by all the models. In the case of batched inference, ENI is performed after an entire batch of nodes has been processed by the ensemble.

5 PRUNING TECHNIQUES

Edge Pruning. The most time-consuming operation in all GNN architectures is the message passing process. For a L-layer GNN, each node receives messages from its L-hop neighborhood, and hence, efficiency is largely dependent on the number of edges in the graph. However, we observe that not all neighbors of a node need to be considered for accurate prediction. In fact, the messages received from many neighbors are redundant, and do not provide any additional useful information for classification. Previous works (Yang et al., 2020; Li et al., 2020; Zheng et al., 2020) have proposed the identification of redundant (and harmful) edges based on network topology and the task at hand, such as task-irrelevant edges and inter-class edges. Complementary to these, Edge Pruning identifies and prunes harmful/redundant/unimportant edges based on the current (trained) state of the GNN. Therefore, the previously proposed techniques are applied pre-training, while Edge Pruning is a post-training optimization that further prunes unimportant edges for highly efficient and accurate inference. Due to the exponential increase in runtime with each edge in the graph, it is vital to find the largest set of edges that can be pruned for a given accuracy loss constraint. In order to identify an optimal set of edges to prune, we need to ensure that when an edge is pruned, the representations of the nodes connected by it remain sufficiently good to maintain correct functionality of the message passing pipeline. We propose a greedy approach to identify the optimal set of edges to prune in Alg. 2, demonstrated on an undirected input graph for clarity. We inspect edges one-by-one, and verify their effect on the two nodes (A and B) connected by them. If an edge is found to have ”minimal” effect on the nodes connected by it, the edge is pruned from the graph, where an approximation knob T_{edge} defines ”minimal” based on the given accuracy constraint. However, since real-world graphs have a large number of edges, this process is computationally expensive. Therefore, in order to avoid inspecting every edge in the graph, we sort the edges in order of a proxy importance score, and inspect only the edges that are likely to be redundant. We establish an importance metric for edges based on three observations: (1) Nodes with high in-degree are likely to have more redundant incoming edges, since messages from several neighbors often contain similar (redundant) information. (2) Nodes that are correctly classified with high confidence are likely to have more redundant incoming edges than low-confidence nodes (which require more information from neighbors for correct classification). (3) Edges with low edge weights are likely to have less effect on the output, and hence, are more likely to be redundant. For an edge connecting two nodes A and B, we compute its importance score as $Importance[edge] = \frac{(ConfidenceA+ConfidenceB)}{2} + \frac{(In\_degreeA+In\_degreeB)}{2\times max\ in\_degree} - \frac{edge\ weight}{max\ edge\ weight}$, where $max\ in\_degree$ and $max\ edge\ weight$ are computed over the entire input graph. We sort edges in descending order of their importance score ($Edgelist$), and inspect them in this order. When M (a small number, typically 3) consecutive edges are identified as not prunable, we stop the Edge Pruning process, since the un-inspected edges are likely to be more important than the edges already inspected, and hence, highly unlikely to be pruned. Hence, the search space and the overheads of Edge Pruning are drastically reduced with minimal impact on efficiency of the final model using importance-ordered inspection of edges with early stopping. In addition, edges in graphs stored by different models in the ensemble, and edges between distinct source-destination pairs in the same graph can be inspected in parallel, further reducing the overheads. We note that for a fair calculation of inference speedup, we only consider edges connecting two labeled nodes or a labeled node with an unlabeled node (considered only in transductive learning tasks; in this case, we check the impact of pruning the edge on the labeled node only. If the edge is identified as unimportant, it is converted to a directed edge from the labeled node to the unlabeled node.).

Network Pruning. The neural networks that process the input graph in GNNs are typically small, and have relatively less impact on the run time compared to the topology of the input graph. However, we find that pruning redundant parameters from the network can still provide tangible improvement in inference efficiency. In this work, we focus on structured pruning that provides speedups irrespective of the underlying hardware. Since ENI reduces the impact of misclassifying one sample on the classification of other samples (bridging the gap between GNNs and other classes of DNNs),

Algorithm 2: Edge Pruning

```
$M$: number of consecutive non-prunable edges required for early stopping
$T_{edge}$: approximation threshold based on acceptable accuracy loss
graph: Undirected input graph; model: GNN trained on graph
$Edgelist$ = $sort\_by\_importance$(edges[graph])
for $each\ Edge\ in\ Edgelist$ do
    $A, B$: Nodes connected by Edge
    $init\_conf\_A$ = classify_node(model, $A$)
    $init\_conf\_B$ = classify_node(model, $B$)
    graph = graph - $Edge$
    $new\_conf\_A$ = classify_node(model, $A$)
    $new\_conf\_B$ = classify_node(model, $B$)
    if $(new\_conf\_A < T_{edge} \times init\_conf\_A)$ and $(new\_conf\_B < T_{edge} \times init\_conf\_B)$ then
        graph = graph + $Edge$
        if $M$ consecutive edges are not prunable then
            return graph
    else if $(new\_conf\_A < T_{edge} \times init\_conf\_A)$ then
        graph = graph + $directedEdge$ from B to A
    else if $(new\_conf\_B < T_{edge} \times init\_conf\_B)$ then
        graph = graph + $directedEdge$ from A to B
return graph
```

and the neural networks used in GNNs are similar to other classes of DNNs, we can make use of the vast literature of works aimed at pruning DNNs. In particular, we use importance estimation using first-order Taylor expansion to rank parameters in order of importance, and then prune the least important ones, following the procedure described in (Molchanov et al., 2017). An approximation knob $T_{network}$ sets the importance threshold (parameters with importance scores below this threshold are pruned) to control the amount of pruning, and $T_{network}$ can be tuned to operate at different points in the accuracy-efficiency curve based on the user's constraints. *Due to the relatively limited impact of Network Pruning on inference efficiency, we find that the optimal solution for a given accuracy constraint involves aggressive tuning of T_{edge} and conservative tuning of $T_{network}$ (such that only highly unimportant parameters are pruned).*

6 Experiments and Results

We implement our techniques using DropEdge (Rong et al., 2020) and DGL (Zheng et al., 2021) in PyTorch. The experiments were performed on a GeForce RTX 2080 Ti GPU with 11GB memory. The validation sets are used for Ensemble Creation, Edge Pruning and Network Pruning, and all results in this section are reported on the test sets. Results in this section are reported with inference batch size of 1, and we demonstrate that our techniques are highly effective for batched inference also (Appendix A). For simplicity, we consider only three different network architectures – GCN (Kipf & Welling, 2017), IncepGCN (Rong et al., 2020) and JKNet (Xu et al., 2018). The best single model is trained using the currently known best hyperparameter values and the best network architecture (from the three listed above) for each task, and it is also used as the base model in the ensemble. The rest of the ensemble is created from the two network architectures not used in the best single model using our Ensemble Creation technique. Additional experiments and results are presented in Appendix A.

GEENIs are faster and more accurate than traditional single models. We present results on Cora, Citeseer and Pubmed datasets (transductive node classification) in semi- and full-supervised settings, and on the large Reddit dataset (inductive node classification) in Table 1. We find that GEENIs outperform the best single model and ensembles without ENI in all tasks, while also being faster. *For Cora (full-supervised), Citeseer (full-supervised) and Pubmed (full-supervised), our best single models match (or are very close to) the current state-of-the-art, and hence, GEENIs advance the state-of-the-art in these tasks, while also being significantly faster than the current state-of-the-art*

Table 1: Results of semi-supervised and full-supervised node classification

Dataset	Best Single Model	Ensemble without ENI	GEENI	Speedup from ENI
Cora (full-supervised)	88.2	88.7	**89.6**	2.1$\times$
Cora (semi-supervised)	83.3	83.7	**85.3**	2.4$\times$
Citeseer (full-supervised)	80.5	81.4	**83.5**	2.3$\times$
Citeseer (semi-supervised)	72.7	73.1	**75.5**	2.6$\times$
Pubmed (full-supervised)	91.6	92.0	**92.8**	2.3$\times$
Pubmed (semi-supervised)	79.5	80.3	**84.1**	2.8$\times$
Reddit	95.42	95.97	**96.72**	2.1$\times$

models. The variation in speedup across different tasks is caused two factors: the number of nodes isolated by ENI (the less accurate semi-supervised models have greater speedup than full-supervised models since more nodes are isolated), and the connectivity of the isolated nodes (isolation of well-connected nodes provides greater speedup).

GEENIs prune better than traditional single models, being significantly faster at iso-accuracy. We prune the models using Edge Pruning and Network Pruning. The amount of pruning is controlled by tuning the approximation knobs T_{edge} and $T_{network}$ to operate at different points on the accuracy-efficiency curve. We find that models pruned with Edge Pruning + Network Pruning are more accurate than those pruned with Universal Graph Sparsification (UGS) (Chen et al., 2021) at the same inference speed for two reasons: (1) By ensuring that node representations of the nodes connected by a pruned edge remain sufficiently good, Edge Pruning finds a larger set of edges to remove for a given accuracy constraint. (2) Tuning T_{edge} more aggressively than $T_{network}$ leads to faster models for a given accuracy constraint compared to jointly pruning both the input graph and the neural networks with equal priority. However, we note that Edge Pruning + Network Pruning adds approximately 10% more post-training overheads than UGS. GEENIs are consistently more accurate than the best single models (Fig. 3), especially under drastic approximations for highly efficient inference. Traditional single models suffer from the propagation of misclassifications, where error nodes have an adverse effect on the classification of all neighboring nodes due to improper message passing. This causes a large drop in accuracy at <10% inference time. In addition, the graph stored by each model in the ensemble can be pruned much more aggressively than the graph stored by the best single model for a given accuracy constraint, since predictions from multiple models are considered for the final prediction. An ensemble of multiple models, with each model processing a smaller, sparser graph is significantly faster than a single model processing a larger, denser graph due to the exponential complexity of message passing. We note that when T_{edge} and $T_{network}$ are set such that only the redundant edges and parameters are pruned, we achieve further speedups of up to 5$\times$ (on top of the speedups from ENI) with no loss in accuracy.

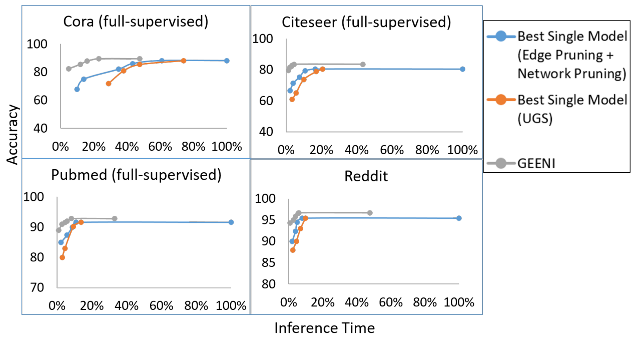

Figure 3: Results of pruning. Edge Pruning and Network Pruning are used, unless specified.

Model diversity is vital for effective ENI. If ENI is performed based on the confidence of the best single model (we modify the confidence threshold T_c, where nodes with confidence less than T_c

are isolated, and report numbers for the maximum test accuracy seen in this process), only 32.7% of error nodes are detected, while 10.4% of correctly classified nodes are falsely identified as error nodes, for a final accuracy of only 79.9%, thereby motivating the use of ensembles for effective ENI. The use of ensembles for ENI is based on the premise that diverse models are unlikely to make the same mistakes. However, creating sufficiently diverse GNNs is challenging, since it has been previously observed that GNN predictions resemble label propagation. This is evidenced by the fact that models created by changing hyperparameters (such as number of GNN layers, hidden size, number of hidden layers etc.) and varying amounts of regularization (DropEdge, Dropout and Sampling) make similar predictions and mistakes (Table 2). This problem persists even when different types of networks are used. However, the use of DropNode (a significantly more drastic approximation), where nodes are randomly dropped from the graph (and the training set), greatly improves diversity, since label propagation is highly dependent on network topology and the training dataset. In addition, the use of Structured Dropout further improves model diversity (Table 2).

Table 2: ENI capability of different ensemble constructs on Citeseer (full-supervised).

Techniques to improve diversity	**Percentage of error nodes detected**	**Percentage of nodes falsely identified as error nodes**	**Accuracy of GEENI**
DropEdge + Dropout + Sampling	9.7	0.4	80.7
+ Network Diversity	16.9	0.5	81
+ DropNode	75.1	0.8	83.2
+ Structured Dropout	**78.8**	**0.8**	**83.5**

Discussion of overheads: In order to create a N network ensemble, our Ensemble Creation method creates one model of the base network (with best known hyperparameters) and K models of every other network, and then constructs the ensemble using models that maximize the Ensemble Quality Score. The use of multiple regularizers, primarily DropNode, greatly speeds up the Ensemble Creation process. For instance, on Citeseer (full-supervised) with IncepGCN, a DropNode rate of 20% makes training a model 9.5x faster on average. We also note that in order to obtain good-performing single-models for any given task, an extensive hyperparameter search needs to be performed. For instance, DropEdge observes that on Citeseer (full-supervised), the difference in accuracy between the best performing 8-layer IncepGCN and the best performing 32-layer GraphSage is 26.9 points. Among the different 8-layer IncepGCN models, we observe accuracy differences of >8 points. By storing the models obtained during this search, the only additional overhead in creating the ensemble involves computing the Ensemble Quality Score with the different models, which is negligible. Our pruning techniques also add small post-training overheads. For each edge inspected during Edge Pruning, we need to classify one or both nodes connected by it in order to ensure their representations remain sufficiently good. The use of importance-based ordering of edges with early stopping reduces the Edge Pruning time by $2-3\times$ over a random exhaustive search of all edges, with negligible impact on inference efficiency. Network Pruning requires a single forward pass through the validation set to characterize importance, and a single iteration of fine-tuning to recover accuracy loss (only under drastic approximations). The wall-clock time for Edge Pruning + Network Pruning is only a few minutes for the smaller graphs (Cora, Citeseer), and approximately 10% of the training time for the larger graphs (Pubmed, Reddit) on a single GPU.

7 Conclusion

We proposed a framework to create Graph neural network Ensembles with ENI (GEENIs) that enables GNNs to enjoy the benefits of ensemble learning while being faster than single models. We introduced an Ensemble Creation method that creates an ensemble of diverse GNNs with the ability to effectively perform Error Node Isolation. We also demonstrated the increased resilience of the GEENI to approximations, and proposed pruning techniques, Edge Pruning (applied to the input graph) and Network Pruning (applied to the neural networks processing the graph), that take advantage of this enhanced resilience. Using this framework, we produced ensemble GNNs that were consistently more accurate and consistently faster than the best-performing single models.

REFERENCES

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rytstxWAW.

Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1695–1706. PMLR, 2021. URL http://proceedings.mlr.press/v139/chen21p.html.

Thomas G. Dietterich. Ensemble methods in machine learning. In Josef Kittler and Fabio Roli (eds.), *Multiple Classifier Systems, First International Workshop, MCS 2000, Cagliari, Italy, June 21-23, 2000, Proceedings*, volume 1857 of *Lecture Notes in Computer Science*, pp. 1–15. Springer, 2000. doi: 10.1007/3-540-45014-9_1. URL https://doi.org/10.1007/3-540-45014-9_1.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SylO2yStDr.

Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/fb4c835feb0a65cc39739320d7a51c02-Abstract.html.

Ben Finkelshtein, Chaim Baskin, Evgenii Zheltonozhskii, and Uri Alon. Single-node attack for fooling graph neural networks. *CoRR*, abs/2011.03574, 2020. URL https://arxiv.org/abs/2011.03574.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 1024–1034, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html.

L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990. doi: 10.1109/34.58871.

Wen-bing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 4563–4572, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/01eee509ee2f68dc6014898c309e86bf-Abstract.html.

Zhihao Jia, Sina Lin, Rex Ying, Jiaxuan You, Jure Leskovec, and Alex Aiken. Redundancy-free computation for graph neural networks. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (eds.), *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pp. 997–1005. ACM, 2020. doi: 10.1145/3394486.3403142. URL https://doi.org/10.1145/3394486.3403142.

Sanjay Kariyappa, Atul Prakash, and Moinuddin K. Qureshi. Protecting dnns from theft using an ensemble of diverse models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=LucJxySuJcE`.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=SJU4ayYgl`.

Edward Elson Kosasih, Joaquín Cabezas, Xavier Sumba, Piotr Bielak, Kamil Tagowski, Kelvin Idanwekhai, Benedict Aaron Tjandra, and Arian Rokkum Jamasb. On graph neural network ensembles for large-scale molecular property prediction. *CoRR*, abs/2106.15529, 2021. URL `https://arxiv.org/abs/2106.15529`.

Jiayu Li, Tianyun Zhang, Hao Tian, Shengmin Jin, Makan Fardad, and Reza Zafarani. SGCN: A graph sparsifier based on graph convolutional networks. In Hady W. Lauw, Raymond Chi-Wing Wong, Alexandros Ntoulas, Ee-Peng Lim, See-Kiong Ng, and Sinno Jialin Pan (eds.), *Advances in Knowledge Discovery and Data Mining - 24th Pacific-Asia Conference, PAKDD 2020, Singapore, May 11-14, 2020, Proceedings, Part I*, volume 12084 of *Lecture Notes in Computer Science*, pp. 275–287. Springer, 2020. doi: 10.1007/978-3-030-47426-3_22. URL `https://doi.org/10.1007/978-3-030-47426-3_22`.

Yanxi Li, Zean Wen, Yunhe Wang, and Chang Xu. One-shot graph neural architecture search with dynamic search space. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 8510–8517. AAAI Press, 2021. URL `https://ojs.aaai.org/index.php/AAAI/article/view/17033`.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=SJGCiw5gl`.

Tianyu Pang, Kun Xu, Chao Du, Ning Chen, and Jun Zhu. Improving adversarial robustness via promoting ensemble diversity. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 4970–4979. PMLR, 2019. URL `http://proceedings.mlr.press/v97/pang19a.html`.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=Hkx1qkrKPr`.

Adria Ruiz and Jakob Verbeek. Anytime inference with distilled hierarchical neural ensembles. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 9463–9471. AAAI Press, 2021. URL `https://ojs.aaai.org/index.php/AAAI/article/view/17140`.

Sanchari Sen, Balaraman Ravindran, and Anand Raghunathan. EMPIR: ensembles of mixed precision deep networks for increased robustness against adversarial attacks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=HJem3yHKwH`.

Thilo Strauss, Markus Hanselmann, Andrej Junginger, and Holger Ulmer. Ensemble methods as a defense to adversarial perturbations against deep neural networks. *CoRR*, abs/1709.03423, 2017. URL `http://arxiv.org/abs/1709.03423`.

Shyam Anil Tailor, Javier Fernández-Marqués, and Nicholas Donald Lane. Degree-quant: Quantization-aware training for graph neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=NSBrFgJAHg`.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

Hongwei Wang and Jure Leskovec. Unifying graph convolutional neural networks and label propagation. *CoRR*, abs/2002.06755, 2020. URL `https://arxiv.org/abs/2002.06755`.

Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=Sklf1yrYDr`.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5449–5458. PMLR, 2018. URL `http://proceedings.mlr.press/v80/xu18c.html`.

Xiaoran Xu, Wei Feng, Yunsheng Jiang, Xiaohui Xie, Zhiqing Sun, and Zhi-Hong Deng. Dynamically pruned message passing networks for large-scale knowledge graph reasoning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=rkeuAhVKvB`.

Bencheng Yan, Chaokun Wang, Gaoyang Guo, and Yunkai Lou. Tinygnn: Learning efficient graph neural networks. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (eds.), *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pp. 1848–1856. ACM, 2020. doi: 10.1145/3394486.3403236. URL `https://doi.org/10.1145/3394486.3403236`.

Han Yang, Xiao Yan, Xinyan Dai, and James Cheng. Self-enhanced GNN: improving graph neural networks using model outputs. *CoRR*, abs/2002.07518, 2020. URL `https://arxiv.org/abs/2002.07518`.

Shaofeng Zhang, Meng Liu, and Junchi Yan. The diversified ensemble neural network. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/b86e8d03fe992d1b0e19656875ee557c-Abstract.html`.

Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. Robust graph representation learning via neural sparsification. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 11458–11468. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/zheng20d.html`.

Da Zheng, Minjie Wang, Quan Gan, Xiang Song, Zheng Zhang, and George Karypis. Scalable graph neural networks with deep graph library. In Liane Lewin-Eytan, David Carmel, Elad Yom-Tov, Eugene Agichtein, and Evgeniy Gabrilovich (eds.), *WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, March 8-12, 2021*, pp. 1141–1142. ACM, 2021. doi: 10.1145/3437963.3441663. URL `https://doi.org/10.1145/3437963.3441663`.

A Additional Results

ENI is highly effective for batched inference. ENI is most effective when a batch size of 1 is used, since error nodes can be immediately identified and isolated. During batched inference, ENI can be performed only after an entire batch of nodes is classified, and hence, error nodes have a detrimental effect on neighbors within the same batch. However, we find that ENI is highly beneficial, even during batched inference (Table 3). We also note that on memory constrained devices such as GPUs, GNNs used for large graphs (such as Reddit) can only use very small batch sizes (typically 4/8) for efficient inference.

Table 3: Effectiveness of ENI for different batch sizes on Citeseer (full-supervised).

Batch Size	**Accuracy of GEENI**	**Speedup from ENI over the best single model**
1	83.5	2.3×
2	83.5	2.3×
4	83.2	2.2×
8	82.9	2×
16	82.5	1.8×
32	81.8	1.7×

3-model ensembles provide best tradeoff between accuracy and efficiency. We experiment with different number of models in the ensemble, and find that 3 models (sufficient to get a majority prediction in most cases) are sufficient for effective ENI. In our experiments, 5-model ensembles (with additional GAT (Velickovic et al., 2018) + GraphSAGE (Hamilton et al., 2017) models) do not improve accuracy over 3-model ensembles. We hypothesize that this is because 3-model ensembles created using our Ensemble Creation method have sufficient diversity to detect most errors made due to the use of approximations for improving training and inference efficiency (such as sampling, pruning etc.), due to randomness (such as parameter initialization, order in which training data is presented etc.) and due to the inherently approximate nature of neural networks. The error nodes not detected by the 3-model ensemble are likely to be ”true outliers”, and GNNs trained using conventional methods are unlikely to predict them correctly. In addition, we find that individual graphs stored by each model in 5-model ensembles cannot be pruned to become sufficiently small and sparse to overcome the overheads of using additional models. Therefore, 3-model ensembles also prune better than the 5-model ensembles.

3-model ensembles add minimal memory overheads. The models used in GNNs tend to be very small (around 5MB) compared to the input graphs (few GBs) in most cases (with some exceptions, such as GNNs used in molecular prediction, where the input graphs are also small). The node features (most memory-consuming part of a graph) are a function of the dataset, and hence, only one copy of the node features needs to be stored. Each model in the ensemble maintains its own individual copy of the adjacency matrix with the appropriate nodes isolated, which is stored in sparse format to minimize memory usage. Hence, the static memory overhead of using additional models in the ensemble is small. On Citeseer (full-supervised), our best single model requires approximately 58MB of static storage for the input graph and the model. Our 3-model ensemble requires approximately 65MB of storage, 12% more than the best single model after Ensemble Creation. After Edge Pruning and Network Pruning with T_{edge} and $T_{network}$ set for no accuracy loss, the ensemble requires only 8% more static memory than the best single model. In addition, the average dynamic GPU memory usage is less for the ensemble than the best single model due to the use of ENI.

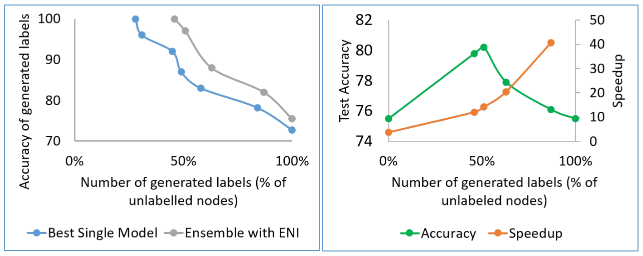

Figure 4: Pseudo-label generation and resulting benefits. Speedups are reported on the nodes that remained unlabeled (nodes that are not part of the training/ validation set, and haven't been added to the list of training nodes as a result of pseudo-label generation), while accuracy is reported over all unlabeled nodes in the original graph (nodes that are not part of the training/ validation set).

GEENIs generate better pseudo-labels than traditional single models. Real world graphs are often sparsely labeled. As a result, some nodes in sparsely labeled neighborhoods are never trained on, resulting in the presence of several error nodes in these neighborhoods. The best known solution to this problem involves the use of pseudo-labels (Yang et al., 2020). Typically, a GNN is first trained on the sparsely labeled graph. Then, this GNN is used to predict the labels of the unlabeled nodes. If the GNN's confidence in predicting a node is above a certain threshold, that node is then added to the training set along with its predicted label, based on the assumption that highly confident predictions are likely to be correct. Then, the GNN is finally trained on this enlarged training set for better generalization performance. Here, we use our ensemble to generate pseudo-labels on Citeseer (semi-supervised), and find that it produces significantly higher quality pseudo-labels than the best single model (Fig. 4). With the tightest constraint, all models in the ensemble must produce the same classification with confidence above a threshold for the node to be added to the training set. We observe that this constraint leads to approximately 50% of the unlabeled nodes being added with 100% accurate training labels. We then iteratively relax the number of models that need to produce the same classification and the confidence threshold to obtain the other points on the curve. Another optimization that takes advantage of pseudo-labels is the deletion of inter-class edges and the addition of intra-class edges to minimize noise and improve message passing in the GNN (Yang et al., 2020). We use the pseudo-labels generated by our ensemble (along with the labels available in the training set) to perform this optimization, and present results in Fig. 4. The peak accuracy on the unlabeled nodes of Citeseer (semi-supervised) is 80.3%, which is 5.4% higher than the peak accuracy of the best single model. With T_{edge} and $T_{network}$ set such that only redundant edges and parameters are pruned with zero accuracy loss, there are two factors that contribute towards increased speedup as more nodes are added to the training set: (1) More inter-class edges can be deleted (based on pseudo-labels), and the resulting benefits far outweigh the overheads of adding a small number of new intra-class edges. (2) More edges can be analyzed and pruned by our Edge Pruning method, since the number of unlabeled nodes decreases (we do not analyze edges connecting two unlabeled nodes).

B Hyperparameters using in Ensemble Creation, Edge Pruning and Network Pruning

We describe the different hyperparameters used in our techniques, and how they can be tuned to operate at different points in the accuracy-efficiency tradeoff space (Table 4).

Table 4: Hyperparameters used in our techniques.

Hyperparameter	**Description**	**Value**
N	Number of models in the ensemble	3
K	Number of models of each network type considered for addition to the ensemble	50 for (Cora, Citeseer, Pubmed), 25 for Reddit due to the compute-intensive training process
M	Number of consecutive non-prunable edges required for early stopping in Edge Pruning	3
T1	Edge Pruning approximation threshold based on accuracy loss constraint	Tuned on the validation set; set such that accuracy on the validation set is as close to the constraint as possible without dropping below the constraint
T2	Network Pruning approximation threshold based on accuracy loss constraint	Tuned on the validation set; set such that accuracy on the validation set is as close to the constraint as possible without dropping below the constraint. T2 is tuned after T1 is set (and the resulting edges pruned) to allow for maximum Edge Pruning