Response to the Reviewers

We thank the reviewers for their critical assessment and insightful comments of our work. We have made extensive modifications to our manuscript. In the following we address their concerns point by point. We also prepare an *annotated version* for our revised manuscript. In the annotated version, changes corresponding to each point are all highlighted by red squares.

Summary of Changes

Reviewer 1

Reviewer Point P 1.1 — The authors conduct an empirical analysis of performance bottlenecks in graph neural network training. The authors identify the edge-related calculation is the performance bottleneck. Experimental of several GNN variants, such as GCN, GGNNNN, GAT and GaAN on six real-world graph datasets verify the importance of the findings. The experimental analysis is sufficient. However, there are some tiny issues in this paper.

Reply: Thank you for your positive comments on our manuscript. We have carefully revised the manuscript according to your comments. Based on the suggestions, we have made an extensive modification on the revised manuscript. Please see the detailed responses below. The changes corresponding to each issue were highlighted by red squares in the annotated version of our manuscript.

Reviewer Point P 1.2 — There are lots of symbols in this paper. Some symbols are reused and confusing, such as s denotes sub-layers or edge features.

Reply: We apologize for the confusing use of symbols. To clarify the symbol usage, we have checked our manuscript and unified the usage of symbol in the revised version. After revision, each symbol only represents one meaning. We added Table 1 at the beginning of Section 2 "Review of Graph Neural Networks" in the revised version to summarize frequently-used symbols. We quote the table below:

Category	Symbol	Meaning
Graph Structure	$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	The simple undirected input graph with the vertex set $\mathcal{V}$ and the edge set $\mathcal{E}$.
	v_x	The x-th vertex of the input graph.
	$e_{y,x}$	The edge pointing from v_y to v_x of the input graph.
	$\mathcal{N}(v_x)$	The adjacency set of v_x in the input graph.
	$\bar{d}$	The average degree of the input graph.
GNN Definition	L	The number of GNN layers.
	K	The number of heads in a GNN layer.
	ϕ^l	The messaging function of the GNN layer l.
	Σ^l	The aggregation function of the GNN layer l.
	γ^l	The vertex updating function of the GNN layer l.
	$\phi^{l,i}$ / $\Sigma^{l,i}$ / $\gamma^{l,i}$	The messaging/aggregation/updating function of the i-th sub-layer of the GNN layer l.
	$\boldsymbol{W}^l, \boldsymbol{W}^{(k)}/\boldsymbol{b}, \boldsymbol{a}$	The matrices/vectors represented by the blue characters are the weight matrices/vectors that need to be learned in the GNN.
Vector	$\boldsymbol{v}_x$	The feature vector of the vertex v_x.
	$\boldsymbol{e}_{y,x}$	The feature vector of the edge $e_{y,x}$.
	$\boldsymbol{h}_x^l$	The input hidden vector of the graph neuron corresponding to v_x in the GNN layer l.
	$\boldsymbol{h}_x^{l+1}$	The output hidden vector of the graph neuron corresponding to v_x in the GNN layer l.
	$\boldsymbol{m}_{y,x}^l$	The message vector of the edge $e_{y,x}$ outputted by ϕ^l of the GNN layer l.
	$\boldsymbol{s}_x^l$	The aggregated vector of the vertex v_x outputted by Σ^l of the GNN layer l.
	$\boldsymbol{h}_x^{l,i}$ / $\boldsymbol{m}_{y,x}^{l,i}$ / $\boldsymbol{s}_x^{l,i}$	The hidden/message/aggregated vector of the vertex v_x outputted by $\gamma^{l,i}$/$\phi^{l,i}$/$\Sigma^{l,i}$ of the i-th sub-layer of the GNN layer l.
	d_{in}^l, d_{out}^l	The dimension of the input/output hidden vectors of the GNN layer l.
	$dim(\boldsymbol{x})$	The dimension of a vector $\boldsymbol{x}$.

In the revised manuscript, we use $e_{y,x}$ to represent an edge and use $\boldsymbol{e}_{y,x}$ to represent its input feature vector. We use $\boldsymbol{s}$ to represent aggregated vectors outputted by the aggregation function Σ in graph neurons. For every vertex v_x, we use $\boldsymbol{s}_x^l$ to denote the aggregated vector in the graph neuron of v_x in the GNN layer l. If the GNN layer l has sub-layers, $\boldsymbol{s}_x^{l,i}$ represents the aggregated vector of v_x in the i-th sub-layer.

To clarify the concept of *sub-layers* in a GNN layer, we have added more description in the revised manuscript. We first introduce the concept of sub-layer in Section 2.2 "Graph Neuron and Message-passing Model" as:

> **Section 2.2 "Graph Neuron and Message-passing Model"**
> Some complex GNNs like GAT [6] and GaAN [7] use more than one message passing phase in each GNN layer. We regard every message passing phase in a GNN layer as a *sub-layer*. We will give out more details on sub-layers when we introduce GAT.

We then use GAT as an example to elaborate on the concept of sub-layers in Section 2.3 "Representative GNNs" as:

> **Section 2.3.3 "GAT (Low Vertex & High Edge Complexity)"**
> Each GAT layer consists of a vertex pre-processing phase and two sub-layers (i.e., message-passing phases).

The vertex pre-processing phase calculates the attention vector $\hat{\boldsymbol{h}}_x^l$ for every vertex v_x by $\hat{\boldsymbol{h}}_x^l = \|_{k=1}^K \boldsymbol{W}_{(k)}^l \boldsymbol{h}_x^l$. We denote the attention sub-vector generated by the k-th head as $\hat{\boldsymbol{h}}_x^l[k] = \boldsymbol{W}_{(k)}^l \boldsymbol{h}_x^l$.
The first sub-layer of GAT (defined in Equation 1) uses the attention vectors to emit the attention weight vector $\boldsymbol{m}_{y,x}^{l,0}$ for every edge $e_{y,x}$ and aggregates the attention weight vectors for every vertex v_x to get the weight sum vector $\boldsymbol{h}_x^{l,0}$.

$$\begin{aligned}
\boldsymbol{m}_{y,x}^{l,0} &= \phi^{l,0}(\boldsymbol{h}_y^l, \boldsymbol{h}_x^l, \boldsymbol{e}_{y,x}, \hat{\boldsymbol{h}}_y^l, \hat{\boldsymbol{h}}_x^l) = \|_{k=1}^K \exp(LeakyReLU(\boldsymbol{a}^T[\hat{\boldsymbol{h}}_y^l[k] \parallel \hat{\boldsymbol{h}}_x^l[k]])), \\
\boldsymbol{s}_x^{l,0} &= \Sigma_{v_y \in \mathcal{N}(v_x)} \boldsymbol{m}_{y,x}^{l,0}, \\
\boldsymbol{h}_x^{l,0} &= \gamma^{l,0}(\boldsymbol{h}_x^l, \boldsymbol{s}_x^{l,0}) = \boldsymbol{s}_x^{l,0}.
\end{aligned} \tag{1}$$

The second sub-layer of GAT (defined in Equation 2) uses the weight sum vectors to normalize the attention weights for every edge and aggregates the attention vectors $\hat{\boldsymbol{h}}_y^l$ with the normalized weights. The aggregated attention vectors $\boldsymbol{s}_x^{l,1}$ are transformed by an activation function δ and are outputted as the hidden vectors of the current layer $\boldsymbol{h}_x^{l+1}$.

$$\begin{aligned}
\boldsymbol{m}_{y,x}^{l,1} &= \phi^{l,1}(\boldsymbol{h}_y^{l,0}, \boldsymbol{h}_x^{l,0}, \boldsymbol{e}_{y,x}, \hat{\boldsymbol{h}}_y^l, \hat{\boldsymbol{h}}_x^l) = \|_{k=1}^K \frac{\exp(LeakyReLU(\boldsymbol{a}^T[\hat{\boldsymbol{h}}_y^l[k] \parallel \hat{\boldsymbol{h}}_x^l[k]]))}{\boldsymbol{h}_x^{l,0}[k]} \hat{\boldsymbol{h}}_y^l[k], \\
\boldsymbol{s}_x^{l,1} &= \Sigma_{v_y \in \mathcal{N}(v_x)} \boldsymbol{m}_{y,x}^{l,1}, \\
\boldsymbol{h}_x^{l+1} = \boldsymbol{h}_x^{l,1} &= \gamma^{l,1}(\boldsymbol{h}^{l,0}, \boldsymbol{s}_x^{l,1}) = \delta(\boldsymbol{s}_x^{l,1}).
\end{aligned} \tag{2}$$

Reviewer Point P 1.3 — Some typical applications of GNNs should be included, such as video object segmentation [ref1], human-object interaction [ref2] and human-parsing [ref3].[1] Zero-shot video object segmentation via attentive graph neural networks,iccv 2019 [2] Learning human-object interactions by graph parsing neural networks, eccv 2018. [3] Hierarchical human parsing with typed part-relation reasoning, cvpr 2020.

Reply: Thank you for pointing out missing references. Computer vision is indeed an important application area of graph neural networks. We have added the mentioned references in Section 1 "Introduction" in the revised manuscript. We quote the related sentence below:

> **Section 1 "Introduction"**
> The powerful expression ability makes GNNs achieve good accuracy in not only graph analytical tasks [8, 9, 10] (like node classification and link prediction) but also computer vision tasks (like human-object interaction [11], human parsing [12], and video object segmentation [13]).

Reviewer Point P 1.4 — There are some grammar errors and typos:
- 'Take the demo GNN in Figure 1(a) as the example.'

- 'to calculate the output hidden vector h^{l+1} of the current layer l, i.e., $h^{l+1} = \gamma^l(h^l, s^l)$ The end-to-end training requires. . . '
- 'Implementing it with the specially optimized basic operators on the GPU is a potential optimization'
- The sentences in the experimental section should be unified.

Reply:

Thank you for pointing them out. We feel really sorry for our carelessness. We have proofread our revised manuscript carefully to eliminate grammar errors and typos. We have also unified the tenses of the sentences in Section 3 "Evaluation Design" and Section 4 "Evaluation Results and Analysis". We use the past tense to describe experimental methods, results and what they indicate. We only use the present tense in the sentences that FigureX/Table X are the subjects of the sentences.

Reviewer Point P 1.5 — Figures 6 and 7 should be adjusted. The figures and fonts are too small.

Reply:

As you suggested, we have adjusted all the figures (besides Figure 6 and Figure 7) in the revised manuscript to make sure that font sizes in figures are no less than the font size of figure captions. To make the figures more easy to read, we also split Figure 6 and Figure 7 into five figures (Figure 6 to 10) in the revised manuscript to enlarge the subfigures.

Reviewer Point P 1.6 — In my view, computation efficiency is to describe the testing or validation process. Except for reporting and analyzing the training times, it is meaningful to discuss the inference time. This is also an important point of view for deep learning researchers to be concerned about.

Reply: Thank you for the insightful comment and suggestion. The efficiency of inference (including inference time and memory usage) is indeed important for deep learning researchers and engineers. In the revised manuscript, we have added a new subsection (Section 2.5 "Inference with GNNs") to briefly review how to perform inference with GNNs. We quote the added subsection below.

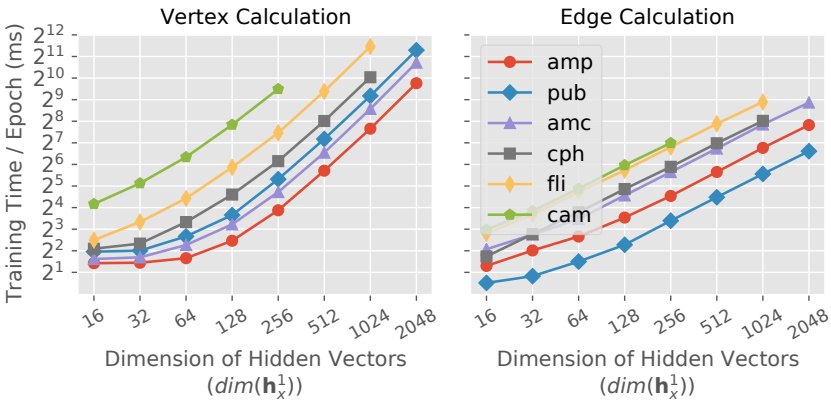
> **Section 2.5 "Inference with GNNs"**
> The model parameters of a GNN consist of the weight vectors/matrices in the prediction layer and the messaging/aggregation/updating functions of each GNN layer. For transductive learning, the input graphs of training and inference are same. The structure of the GNN does not need to adjust. For inductive learning, the input graphs used in training and inference are different. When we want to perform inference on a new graph with a pre-trained GNN, the structure of the GNN needs to be adjusted, but the hyper-parameters and the model parameters of the GNN remain unchanged. The number of graph neurons in each GNN layer has to be adjusted to $|\mathcal{V}|$ of the new graph. The connections of graph neurons between GNN layers are also adjusted according to the edge set of the new graph. During inference, input feature vectors of the input graph

> are propagated forward from the input layer to the prediction layer to give out predicted labels.
> When the memory capacity of the GPU is large enough to hold all of the input graph and intermediate results during inference, the inference can produce predicted labels for all vertices and edges at once, making full use of the computing power of the GPU. When the memory capacity is not large enough, the sampling technique must be adopted.
> The sample-based inference splits the test set into several batches. For each batch, the sample-based inference samples a subgraph related to the batch from the input graph. It performs inference on the sampled subgraph to produce predicted labels for the batch. Since the sampled subgraph is usually much smaller than the original input graph, the memory usage during inference is limited. The sampling method depends on the prediction task. Taking the node classification task as an example, if there are L GNN layers in the GNN, the hidden vector outputted by the last GNN layer of each vertex $\boldsymbol{h}_x^L$ is only related to the L-hop neighborhood of v_x. Thus, if we want to predict labels for a group of vertices, we only need to sample a subgraph that contains L-hop neighborhoods of all given vertices.
> The implementation of sample-based inference in PyG stores the whole input graph and its feature vectors in the main memory. In the node classification task, it splits the vertices in the test set into batches accoding to a given batch size. The subgraph corresponding to each batch is sampled on the CPU side and sent to the GPU side to perform inference.

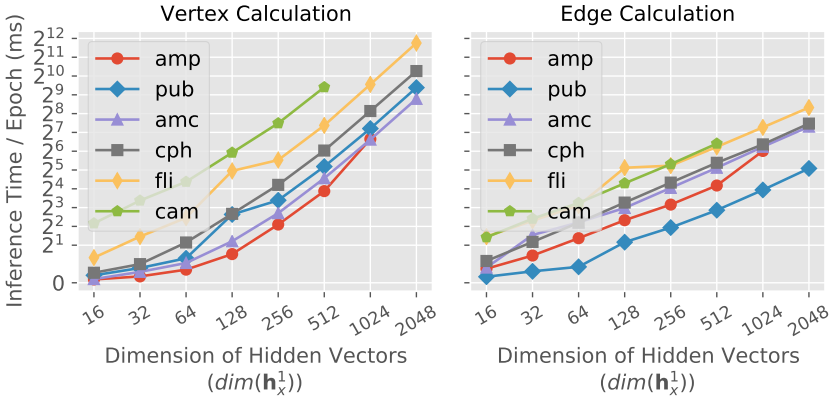
To discuss the efficiency of GNN inference on GPUs, we have added corresponding analysis in every subsection of Section 4 "Evaluation Results and Analysis". Below, we introduce our main findings subsection by subsection.

Section 4.1 "Effects of Hyper-parameters on Performance" We added paragraphs "Effects on Inference Time" in the middle of Section 4.1 to discuss how the time and the peak memory usage of inference changed as we increased the values of the hyper-parameters.

For all GNNs, we found that the effects of hyper-parameters on the *inference time* were the same as the training time. Taking GGNN as an example, Figure 1 in the response (Figure X and Figure X in the revised manuscript) shows the effects of the hidden vector dimension $dim(\boldsymbol{h}_x^1)$ on training/inference time. When $dim(\boldsymbol{h}_x^1)$ was large enough, the vertex/edge calculation time of training time and inference time both grew linearly with $dim(\boldsymbol{h}_x^1)$. We observed similar phenomena in the other GNNs. The results indicated that the time complexity analysis of GNNs were applicable to both training and inference.



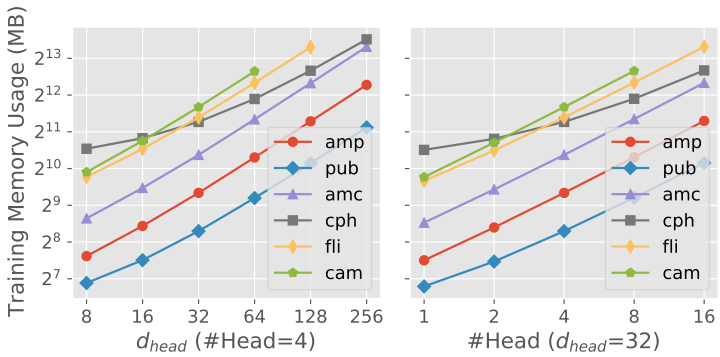
(a) Training



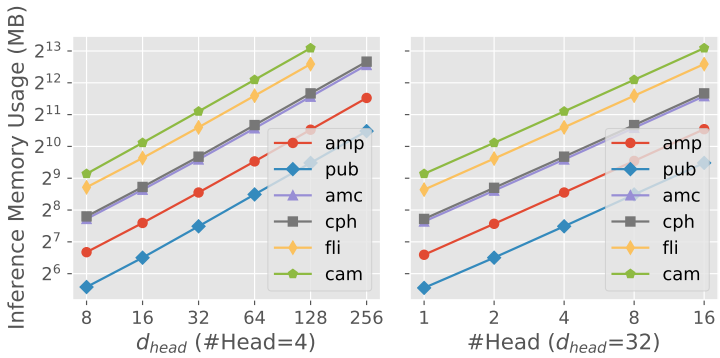
(b) Inference

Figure 1: Effects of hyper-parameters on the vertex/edge calculation time of GGNN.

The effects of hyper-parameters on inference memory usage were also same as training. Taking GAT as an example, Figure 2 in the response (Figure X and Figure X in the revised manuscript) shows the peak memory usage during training and inference under different values of hyper-parameters. The trends of the curves in Figure 2a and Figure 2b were highly similar. The memory usage of training and inference *both* grew linearly as the dimension of each head and the number of heads increased. We observed similar phenomena in the other GNNs.



(a) Training

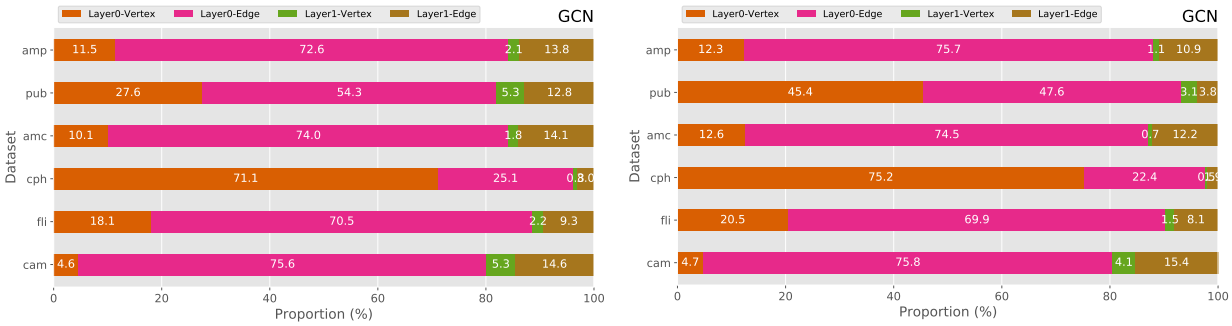


(b) Inference

Figure 2: Effects of hyper-parameters on peak memory usage of GAT.

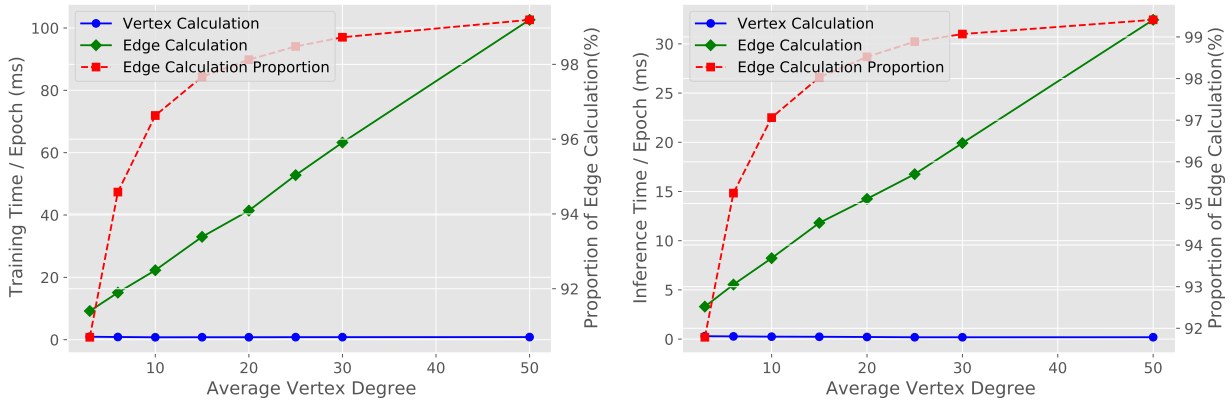
Section 4.2 "Time Breakdown Analysis" We have added a new subsection Section 4.2.4 "Performance Bottlenecks in Inference" in Section 4.2 to conduct time breakdown analysis for GNN inference and compare the differences between training and inference.

We found that the performance characteristics of GNN inference were highly similar to training on the *layer* level and the *step* level. In this reply, we used GCN as an example. We made similar observations in the other GNNs. Figure 3 in the response compares the time breakdowns of GCN on the layer level of training and inference. The time breakdowns were very similar. Figure 4 in the response compares the effects of the average degree on the edge/vertex calculation time during training and inference. The curves in the two sub-figures of Figure 4 have similar trends. *The edge calculation stage dominated both the training and inference time*. Figure 5 in the response further compares the time breakdowns on the step level of the edge calculation stage. The proportion of each step was very similar between training and inference. The results indicated that *the performance bottlenecks of training and inference were same on the layer level and step level*.



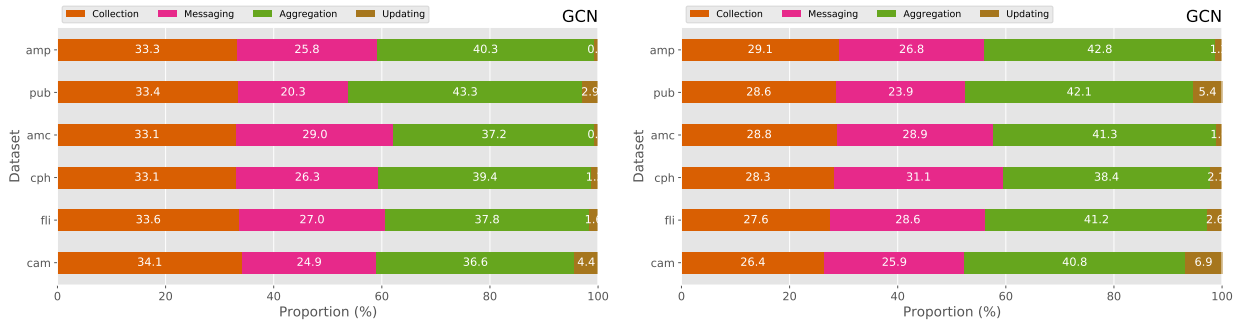
(a) Training (b) Inference

Figure 3: Time breakdowns on the layer level of GCN.



(a) Training (b) Inference

Figure 4: Effects of the average degree on the edge/vertex calculation time of GCN. Graphs were generated by fixing the number of vertices as 50,000.

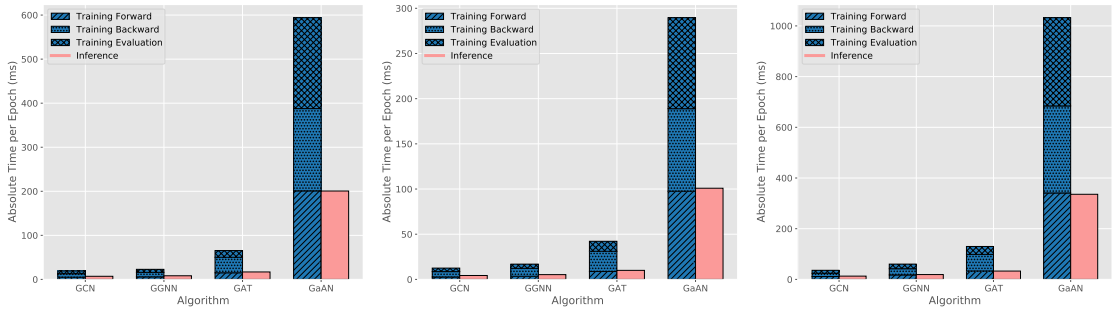


(a) Training (b) Inference

Figure 5: Time breakdowns on the step level of the edge calculation stage of GCN.

The main differences between training and inference were reflected in two aspects: the wall-clock time and the top time-consuming basic operators.

The inference time was much less than the training time. Figure 6 in the response (Figure X in the revised manuscript) compares the wall-clock time of training and inference on the `amp`, `amc`, and `fli` datasets. The results on the other datasets were similar. Since the inference only conducted the forward propagation from the input layer to the prediction layer, the time of inference was very close to the time of the forward phase in training. The inference time was 34% (GCN), 32% (GGNN), 25% (GAT), and 32% (GaAN) of the training time, averaged over all datasets.



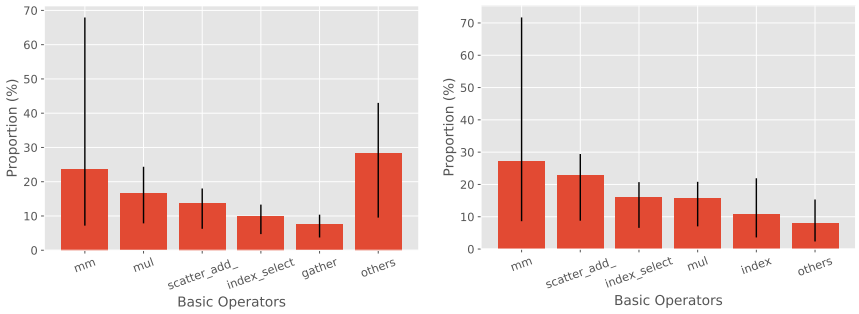
(a) `amc` (b) `amp` (c) `fli`

Figure 6: Wall-clock training/inference time on different datasets.

The top time-consuming basic operators of training and inference also showed a certain degree of difference. Some of the top time-consuming operators during training were replaced by new operators in inference. Figure 7 in the response (Figure X in the revised manuscript) compares the top 5 time-consuming basic operators in training and inference. For GCN, the `index` operator used in the prediction layer became the new top 5 time-consuming operator, replacing the `gather` operator used in the backward phase in training. For GGNN, `index` operator in the prediction layer also became one of the top 5 time-consuming basic operators. For GAT, the `input_put_impl`

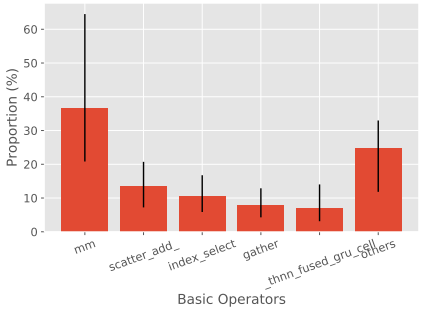
operator (used in the backward phase) in Figure 7e was replaced by the `scatter_add` operator used in the forward edge calculation in Figure 7f. For GaAN, the `mm` operator was replaced by the `cat` operator used in the vertex updating function.

Although some specific time-consuming operators were different, the performance bottlenecks on the operator level were same for training and inference. The matrix multiplication `mm` and the element-wise multiplication `mul` operators were still time-consuming for inference, making GNN inference also suitable for GPUs. The basic operators related to the collection and aggregation steps in the edge calculation still consumed non-trivial time in inference.

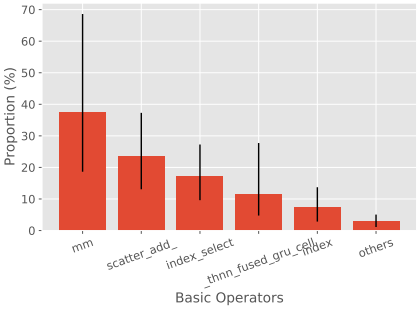


(a) GCN, Training

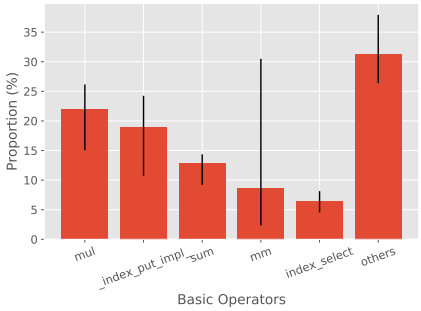
(b) GCN, Inference



(c) GGNN, Training

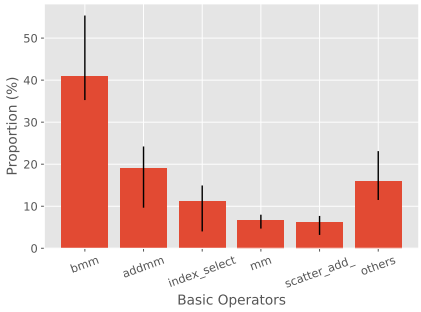


(d) GGNN, Inference

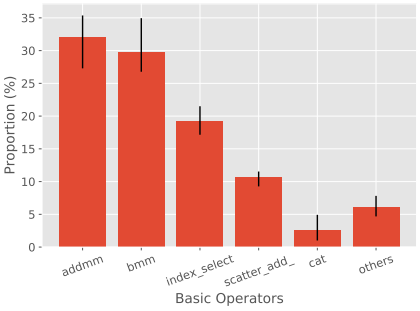


(e) GAT, Training

(f) GAT, Inference



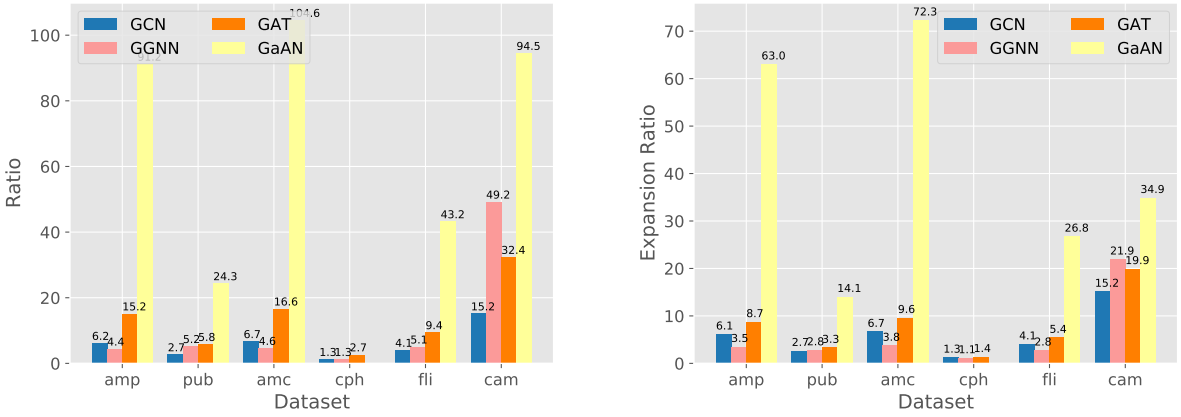
(g) GaAN, Training



(h) GaAN, Inference

Figure 7: Top 5 time-consuming basic operators of typical GNNs. The time proportion of each basic operator was averaged over all datasets with the error bar indicating the maximum and the minimum.

Section 4.3 "Memory Usage Analysis" In the end of Section 4.3, we added a new paragraph "Memory Usage of Inference" to compare the memory usage of inference and training. Figure 8 in the response (Figure X in the revised manuscript) compares the memory expansion ratios of typical GNNs during training and inference. Since no intermediate results had to be cached during inference of GGNN, GAT, and GaAN, the memory expansion ratios of inference were much less than training for the three GNNs. The MERs of inference were 45% to 83% (GGNN), 52% to 61% (GAT), and 37% to 69% (GaAN) of training. However, the MERs were still high, disallowing inferencing with big graphs. To handle big graphs, sample-based inference was necessary.



(a) Training (b) Inference

Figure 8: Memory expansion ratios of typical GNNs.

Section 4.4 "Effects of Sampling Techniques on Performance" In Section 4.4, we added a new subsection Section 4.4.4 "Performance Bottlenecks in Inference" to discuss the performance bottlenecks in sample-based inference.

Sample-based inference was an effective technique to handle big graphs. Taking the node classification task as an example, to predict labels for a given set of vertices $\mathcal{V}_{predict}$, the inference sampler sampled a subgraph containing complete L-hop neighborhoods of all vertices in $\mathcal{V}_{predict}$, where L was the number of GNN layers. Since the real-world graphs often had small-world property, the size of the sampled subgraph increased quickly as the number of vertices in $\mathcal{V}_{predict}$ (i.e., batch size) increased. Figure 9 (Figure X in the revised manuscript) shows how the relative batch size ($\frac{|\mathcal{V}_{predict}|}{|\mathcal{V}|}$) affected the average degree and the number of edges of the sampled subgraphs during inference. When the relative batch size was 10%, the number of edges in the sampled subgraphs were close to the number of edges in the whole graph. In order to limit the memory usage during inference, the batch size used in inference should be very small.

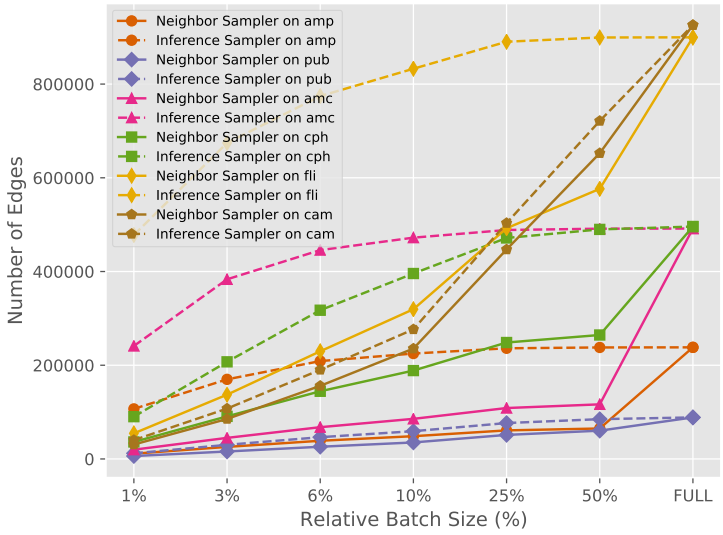
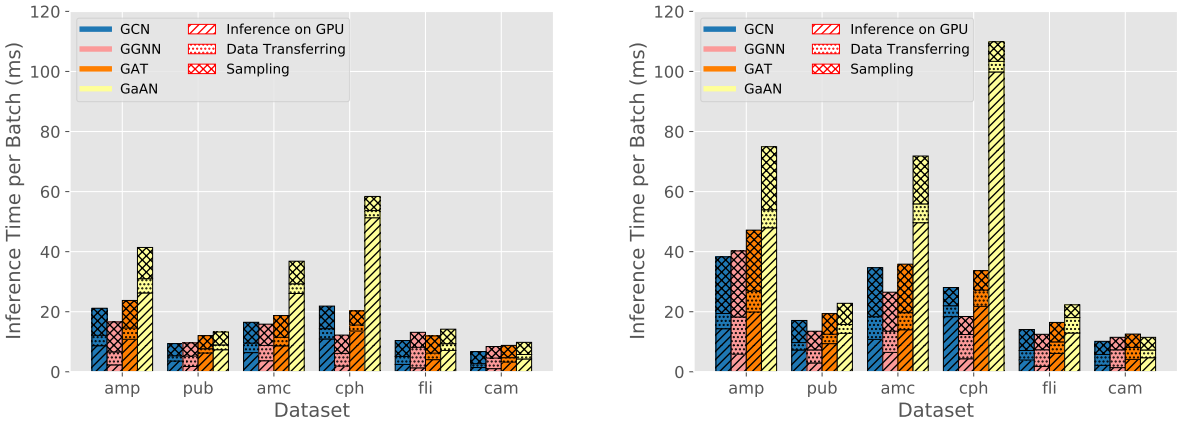


Figure 9: Number of edges in the sampled subgraphs produced by the inference sampler under different relative batch sizes.

When the batch size was small, the overheads brought by sampling and data transfering from CPU to GPU became obvious. Figure 10 (Figure X in the revised manuscript) shows the time breakdown of sample-based inference on different datasets. On the `amp` and `cam` datasets, the sampling time could account for near half of the total inference time. The current implementation of inference sampling in PyG was also inefficient. Improving its efficiency could significantly reduce the inference time.



(a) Batch Size = 1024

(b) Batch Size = 2048

Figure 10: Inference time per batch breakdown under different batch sizes.

Reviewer 2

Reviewer Point P 2.1 — In the paper, authors accomplished a unique study and analysis on GNN models training complexity. The articles first review and development history of GNNs and creatively model all architectures as input layers, intermediate layers of graph neurons and prediction layers. And they quantitatively summarize the time and space complexity of 4 representative GNNs, including graph convolution, gated recurrent graph net, graph attention net and GraphSage. Most importantly, the article first break down complexity into operator level and offered analysis of good granularity, giving reader more guidance in future study. At last, the solid experiments included the study of effects of hyper-parameters and a comparison of two major sampling techniques: neighbor sampling and cluster sampling.

Reply: Thank you for your positive comments on our manuscript. We have carefully revised the manuscript according to your comments. We have revised our manuscript according to your kindly suggestions. Please see the detailed responses below. We have highlighted our modification point by point in the annotated version of the manuscript by red squares.

Reviewer Point P 2.2 — In general, the paper was well written and organized with good structure and clear narratives. Just some minor language errors like line Page 8, Line 208, "In active graph neurons" =>"Inactive graph neurons".

Reply: Thank you for pointing them out. We feel really sorry for our carelessness. We have proofread our revised manuscript carefully to eliminate such language errors as much as we can.

Reviewer Point P 2.3 — I was impressed by the way that authors categorize layers and operators in GNNs, very clear and instructive.

It is also pretty neat to divide layer time complexity into two buckets: vertex calculation and edge calculation. The data model pretty well summarizes mainstream GNN layer architectures. And this analysis is very insightful for layer profiling.

And the experimental evaluation were done over 6 large graph-structured datasets.

Reply: Thank you very much for your appreciation.

Reviewer Point P 2.4 — While, one major drawback is that I did not clearly see the analysis complexity v.s. accuracy. For example, in Figure 19 and 20, I did not see network accuracy from those 4 GNNs. There is always tradeoff between model complexity and model performance, and in some scenarios where high complexity is allowed, a sophisticated model of more powerful representation capability is still needed.

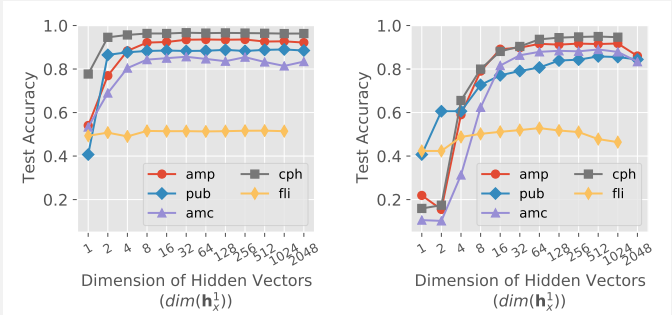
Reply: Thanks very much for your valuable suggestions. The model complexity directly affected both the accuracy and the training time. In order to analyze the relationship between model complexity and accuracy, we conducted two kinds of extra experiments in the revised manuscript: (1) how the hyper-parameters of the GNNs (like the dimension of hidden vectors and the number of

heads) affected the accuracy of GNNs (in Section 4.1); (2) how the batch size in the sampling methods affected the accuracy of GNNs (in Section 4.4).

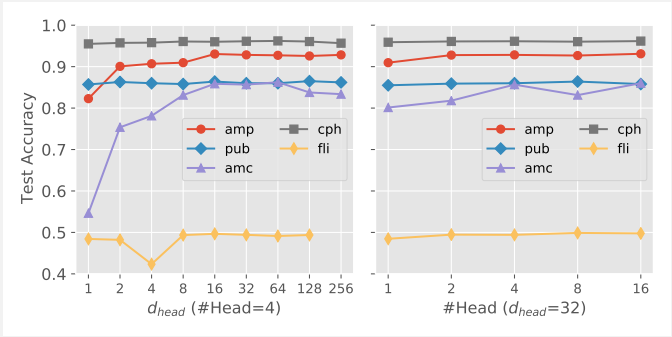
In this reply, we focus on the first kind of experiments added in Section 4.1. We will introduce our results of the second kind of experiments in the next reply.

Hyper-parameters determined the model complexity of a GNN. Generally speaking, higher values of hyper-parameters brought higher model complexity, affecting model accuracy. In the revised manuscript, we added extra paragraphs "Effects on Accuracy" at the end of Section 4.1 "Effects of Hyper-parameters on Performance" to analyze how the hyper-parameters affect the accuracy. We had two main findings. First, the accuracy of GNNs was much more sensitive to the dimension of hidden vectors $dim(\boldsymbol{h}_x^1)$ (for GCN/GGNN/GaAN) and the dimension of each head d_{head} (for GAT) than the other hyper-parameters. Second, the relative accuracy between the four typical GNNs varied greatly with different datasets. We quote the related paragraphs from the revised manuscript below.

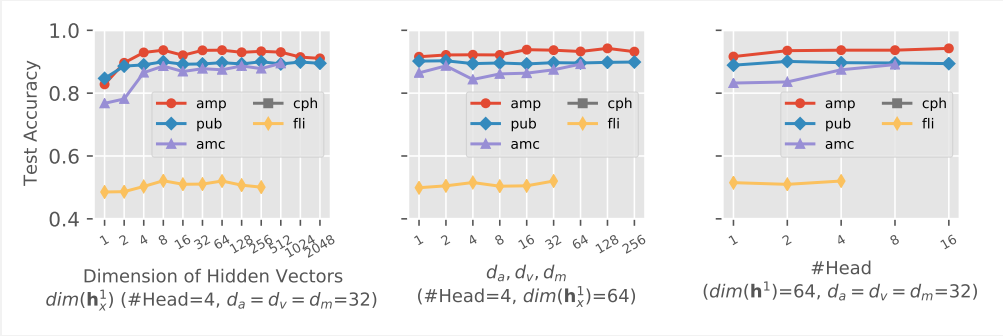
> **Section 4.1 "Effects of Hyper-parameters on Performance"**
> *Effects on Accuracy*. The relationship between hyper-parameters of GNNs (i.e., model complexity) and accuracy is very complicated, and there is no clear theoretical analysis yet. Generally speaking, higher model complexity can bring powerful representation capability and may bring higher accuracy, but it also increases the risk of overfitting. To evaluate the effects of hyper-parameters on accuracy, we measured the accuracy of the four typical GNNs under different hyper-parameters. Figure 11 shows the experimental results.
> For GCN, the accuracy was sensitive to the dimension of hidden vectors $dim(\boldsymbol{h}_x^1)$. As $dim(\boldsymbol{h}_x^1)$ increased, the accuracy first increased quickly and then stabilized when $dim(\boldsymbol{h}_x^1) \geq 8$. For GGNN, its accuracy curves showed similar trends as GCN. However, GGNN was more sensitive to $dim(\boldsymbol{h}_x^1)$ than GCN. Its accuracy even decreased when $dim(\boldsymbol{h}_x^1) \geq 1024$. Since GGNN had high model complexity (with 13 weight matrices/vectors to train), GGNN might occur overfitting in those cases. For GAT, its accuracy was more sensitive to the dimension of each head d_{head} than the number of heads. For GaAN, only $dim(\boldsymbol{h}_x^1)$ showed obvious impacts on accuracy. The experimental results indicated the accuracy of the GNNs was often low when $dim(\boldsymbol{h}_x^1)$ (for GCN/GGNN/GaAN) or d_{head} (for GAT) was small. In those cases, the low model complexity limited the expressive power of the GNNs. As $dim(\boldsymbol{h}_x^1)$ or d_{head} increased to a certain threshold, the GNNs had sufficient learning ability to achieve stable accuracy.



(a) GCN (b) GGNN



(c) GAT



(d) GaAN

Figure 11: Effects of hyper-parameters on accuracy of the typical GNNs.

Figure 12 compares the best accuracy that each GNN achieved on different datasets. The best accuracy that the four GNNs achieved was very close. It was also close to the accuracy reported in their original references [1][4][6][7]. There was no clear winner in terms of accuracy. The relative accuracy between GNNs varied greatly with different datasets. GaAN achieved the highest accuracy in three out of five datasets. GCN also achieved the highest or second highest accuracy in three out of five datasets, though its model was simplest. It indicated that simple GNN models (such as GCN) could still

achieve good accuracy with proper hyper-parameter settings.

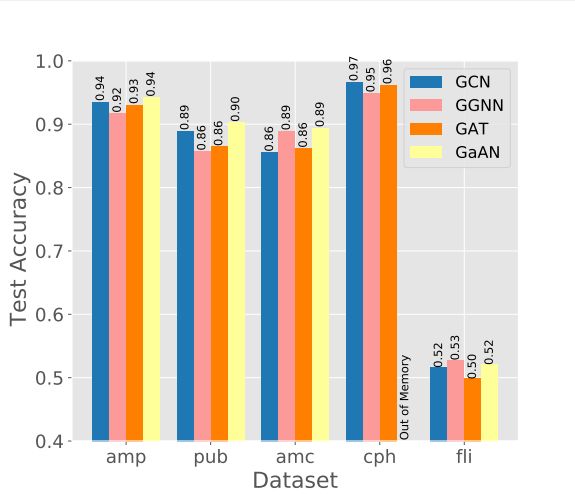


Figure 12: Best accuracy that each GNN achieved on different datasets.

Reviewer Point P 2.5 — Sampling method is definitely going to reduce model complexity, since all models complexity depend on graph node number N, while performance is going to be compromised as well. I would like to see authors resolve the concern of significant accuracy drop after applying aggressive sampling of subgraphs.

Reply:

Thank you for your insightful comment and valuable suggestion.

Reviewer Point P 2.6 — Hope authors supplement the effect of sampling and GNNs on accuracy while comparing different complexity of model and sampling methods.

Reply: