A. MODEL SUMMARY

A1. Background on you/your team

Competition Name: predict-ai-model-runtime
(https://www.kaggle.com/competitions/predict-ai-model-runtime)
Team Name: none
Private Leaderboard Score: 0.70549
Private Leaderboard Place: rank 6

A2. Background on you/your team

What is your academic/professional background?
I am a contract computer vision and deep learning algorithm engineer. My job includes discovering fracture in x-ray images and implementing visual slam for robotic navigation.

Did you have any prior experience that helped you succeed in this competition?
I am familiar deep learning and build deep models in my work. I also have experiences in graph neural net GNN from previous Kaggle competitions.

What made you decide to enter this competition?
The problem is interesting and I can learn from other kagglers about the latest AI methods and insights.

How much time did you spend on the competition?
About 9 hours per day, over two weeks.

A3. Summary

In this competition, we are asked to predict runtime of input TPU computational tensor graphs. For layout runtime prediction, see figure.1:

- reduce input graph to subgraph by including only the 5-hop neighbours from the configure node
- design a GNN model with 4-layer SAGE-conv[2] with residual shortcut
- design a GNN model with 4-layer GIN-conv[3]
- use graph instance normalisation over node [1], for the two models above

For tile runtime prediction, see figure.2:
- design a GNN model with 4-layer GAT-conv[4]

All models are build with python and pytorch deep learning framework. Specifically, we use pytorch geometric[5] for GNN.

Training takes abopout 4 to 6 hours for one model in each collection, using Nvidia Quadro RTX 8000 GPU with memory 48 GB.

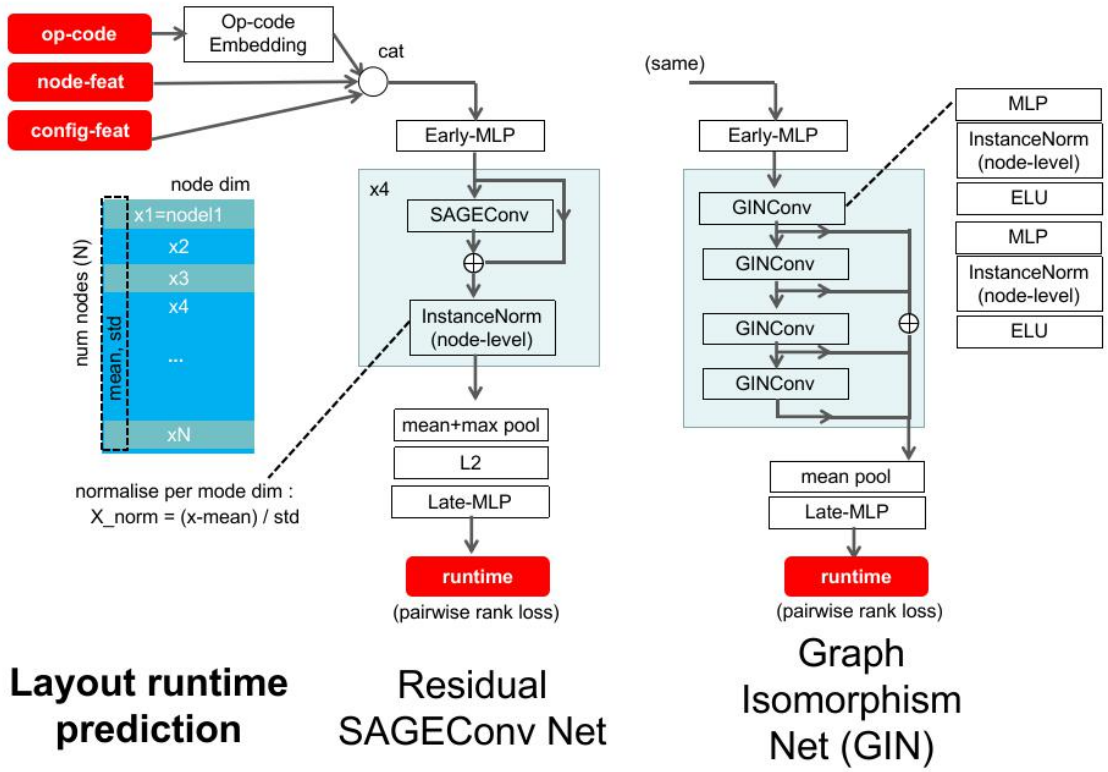


Figure.1 : Layout runtime prediction

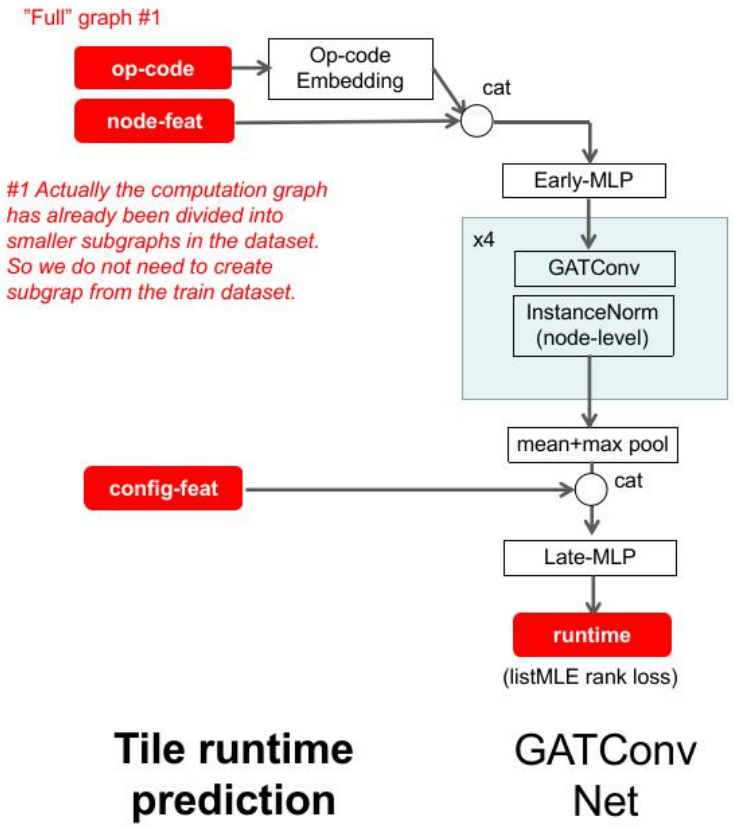


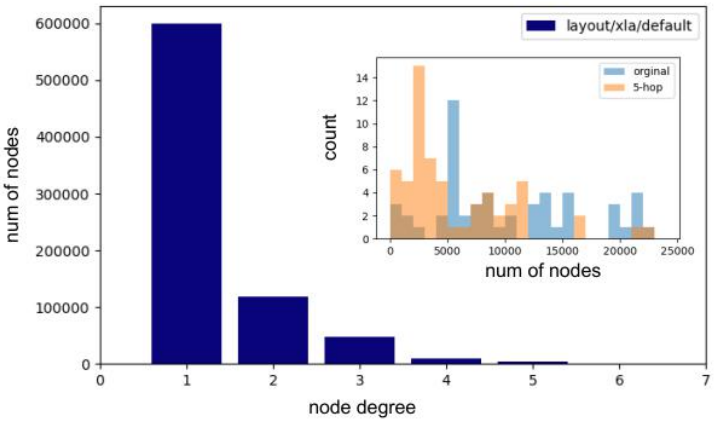
Figure.2 : Tile runtime prediction

A4. Features Selection / Engineering

One issue in the competition is the large size of the graph input data. The competition data, TPUGraphs is one of the largest public graph dataset. The main layout subset has 100 millions graphs of 10 thousands nodes.

We need to use the full graph as our target is computation graph runtime prediction, a graph-level property. But we are interested in the relative runtime ranking, hence we just need to consider the "difference of two graph" to predict the "difference in runtime".

We propsed to select nodes that are 5 hops from the configure nodes to reduce the graph size.



		XLA/Layout Default
graphsage GNN		kendall tau
	3-hop	0.41260
	4-hop	0.42503
	5-hop	0.45508
	6-hop	0.43508

Figure.3 : 5-hop neighbourhood subgraph

For example, in figure.3, for collection layout/xla/default, about 99% of train graph nodes have degree (num of neighours) equal to 5 or less. 5-hop neighbourhood subgraph reduce number of nodes per graph from 13000 ± 5700 to 10000 ± 5000. This is a 25% reduction. Further, from the performance metric (kendall tau) of SAGE-conv GNN, 5-hop is optimal.

A5. Training Method(s)

To train the GNN models, we perform back propagation with ADAM optimizer. We use fixed learning rate of 0.0005 for 2 to 5 epoches. To prevent overfitting and improve generalisation, we use the following:

1. Instead of selecting the best weight at training, we use stochastic weight averaging SWA, to average the last 10 weights. This has improvement of about 0.01 in kendall tau validation.

2. We use gradient accumulation in each training iteration. Batch size is about 32 subgraph. For each subgraph, we random sample about 80 to 100 configurations for ranking loss.

```
optimizer.zero_grad()

for b in range(batch_size):
        r = batch[r]
        loss = net(r)  # forward one subgraph
        scaler.scale(loss).backward() #backward accumuate gradient

scaler.step(optimizer) #update net parameters
scaler.update()
```

Figure.4 : Gradient accumulation in training

Pairwise rank loss is used to train layout prediction model to maximise kendall tau metric:

$$\text{loss}(r1,r2) = -\begin{cases} \log(\text{sigmoid}(r1-r2)) & \text{if } t1 \geq t2 \\ \log(\text{sigmoid}(r2-r1)) & t2 > t1 \end{cases}$$

where r and t are the predicted and ground truth runtime respectively.

ListMLE loss is use for tile prediction instead. Here are are interested in the top5 prediction in the slowdown metric.

Due to the lack of time, we did not experimet much with ensembling. In Figure.5, the ensemble of SAGEconv GNN and GINcov GNN shows improvement for collection layout/xla/default.

		collection				
		NLP/Layout		XLA/Layout		XLA/Tile
		Default	Random	Default	Random	
[a]	4x-gatconv-listmle					0.97462
[b]	4x-graphsage-pair2	0.53938	0.92654	0.45508	0.67128	
[c]	4x-gin-pair2			0.45958		
[b]+[c]	ensemble			0.46952		
submission		0.53938	0.92654	0.46952	0.67128	0.97462
					avg	0.71627
					public lb	0.69424
					private lb	0.70549
For GIN Net, we don't have time to train for all layout prediction before the competition ends.						

Figure.5 : local validation and public/private leaderboard scores

A6. Interesting findings

What was the most important trick you used?
Graph instance nornalisation[1] was the most import trick used. It results in faster convergence in training and improve validation kendall tau by 0.02. The normalisation is also suitable for gradient accumulation as it does not use batch statistics.

Also 5-hop neighbourhood graph reduction enables training with large number configuration sampling per graph. We use 80 to 100 configurations per graph for a single Nvidia Quadro RTX 8000 GPU with 48 GB memory. We find that training is sensitive to batch size and number of configuration sampling. In general, larger number of configuration sampling gives better results.

What do you think set you apart from others in the competition?
There are only a few graph in validation and test set. We note that validation kendall tau can be quite different for each different sample, i.e. high std variance. Hence when selecting the final submission, one should select a solution with both good mean and std validation values.

A7. Simple Features and Methods

Many customers are happy to trade off model performance for simplicity. With this in mind:
Is there a subset of features that would get 90-95% of your final performance?

From our experiment in figure.3, you can use 4-hop subgraph instead.

A8. Model Execution Time

How long does it take to train your model?
Training takes abopout 4 to 6 hours per model for each collection, using a Nvidia Quadro RTX 8000 GPU with 48 GB memory.

How long does it take to generate predictions using your model?
The test data with 8 to 17 graphs, each having 1000 configurations, takes about one min to complete. The validation data having 10000 configurations takes about 5 to 8 min.

How long does it take to train the simplified model (referenced in section A7)?
I expect a reduction of about 25% in time.

How long does it take to generate predictions from the simplified model?
I expect a reduction of about 25% in time.

A9. References

[1] “Learning Graph Normalization for Graph Neural Networks” - Yihao Chen
https://arxiv.org/abs/2009.11746

[2] “Inductive Representation Learning on Large Graphs” - William L. Hamilton
https://arxiv.org/abs/1706.02216

[3] “How Powerful are Graph Neural Networks?” - Keyulu Xu
https://arxiv.org/abs/1810.00826

[4] “Graph Attention Networks” - Petar Veličković
https://arxiv.org/abs/1710.10903

[5] PyG (PyTorch Geometric)
https://github.com/pyg-team/pytorch_geometric