AI Runtime Forecasting: Prediction Using GNNs

Harsha Karanth, Rohit Arunachalam, Anvitha Veeragandham & Shreya Santhanagopalan
{hkaranth3, rarunachalam, aveeragandham3, ssanthan3} @gatech.edu

Group ID: 3

ABSTRACT

As the field of ML grows, more and more models are productionized and used by consumers. Especially with the advent of LLMs, it is a very important task to understand and estimate the amount of time a given model will take to run. Google is offering $50,000 to the group that creates the most successful model runtime predictor - which goes to show just how sought after. Neural networks can be represented by a graph. Each node is an operation (e.g. matrix multiplication, convolution, etc.), and an edge represents the tensor being sent from (u, v) where u and v are both operations (nodes). Nodes, edges, possible configuration, and the assoicated runtime are given in the dataset. We compare architectures (NN vs GNN) on the problem and confirm the superiority of the GNN on this problem. Further, we experiment and iteratively improve our GNN model and eventually create a Sage Conv GNN for this task. The normal/baseline GNN contains embedding layers, graph convolutional layers with messaging passing, and etc. while the Sage Conv GNN contains Sage Conv layers in lieu of the convolutional layers. These layers aggregate complex features from a sample of extended neighboring nodes and improve performance. We used the Kendall Tau Correlation to measure the model's performance, and we discovered that the linear regression performed worse when compared to the GNNs and that the Sage Conv layers improved the ranking score.

INTRODUCTION

Accurate runtime forecasting for AI models is an essential yet challenging task in the efficient deployment of machine learning systems. This project advances the domain of computational performance prediction by leveraging the representational potential of Graph Neural Networks (GNNs). We model AI architectures as directed acyclic graphs where nodes denote computational operations, and edges represent data dependencies via tensors, following the TensorFlow (TF) execution paradigm.

Utilizing the TpuGraphs dataset, which encompasses performance metrics of XLA-compiled graphs on Tensor Processing Units, we experiment with various GNN architectures. We also run the input data into a vanilla NN baseline and compare against a baseline GNN to establish and confirm the superiority of the GNN on this sort of input data. A baseline GNN is first established, employing an embedding layer to initialize node representations. We then extend this with a Sage GNN model, incorporating GraphSAGE's neighborhood aggregation strategy and cross-attention mechanisms to refine the embedding space with respect to the graph topology and execution dynamics.

Our experimental results demonstrate that the Sage GNN, with its advanced feature aggregation and node embedding techniques, yields a superior Kendall Tau Correlation metric when compared to the baseline GNN. This metric reflects the model's capacity to mirror the true runtime distribution of AI models. Furthermore, we address computational bottlenecks associated with NPZ dataset complexity, lengthy training durations.

The study concludes that the application of SageConv layers, coupled with strategic dropout and non-linear activation functions, significantly enhances runtime prediction accuracy. The introduction of cross-attention is posited as a potential avenue for capturing inter-node operational dynamics that are otherwise obfuscated in standard GNN approaches. Future work will focus on optimizing these cross-attention SageConv GNN architectures and experimenting with various loss functions, such as Hinge Loss, to further push the boundaries of runtime predictive accuracy in AI systems.

.

DATA

Our project operates on the TpuGraphs dataset, a specialized collection of performance prediction data on XLA High-Level Optimizer (HLO) graphs executed on Google's Tensor Processing Units (TPUs). This dataset is instrumental in modeling the runtime of AI operations, as it captures the intricate dynamics of computational graphs tailored for machine learning tasks. The data originates from a diverse range of ML models

compiled into HLO instructions for TPUs, offering a detailed view of execution characteristics.

The TpuGraphs dataset is composed of NPZ files—a format leveraging the NumPy library to store arrays of variable sizes, including metadata. Each file represents a distinct computational graph with the following features. Nodes that correspond to AI operations such as matrix multiplications, activations, or convolutions, each encoded with unique operation codes. We have directed edges that represent the data flow from one operation to another and possible optimization configurations for each node, affecting runtime. We also have a runtime label which is the actual execution time of the graph, serving as the target for our predictive models.

In total, the dataset includes thousands of graph instances across five distinct collections. These collections vary by layout configurations and optimization strategies, such as random layouts, default layouts, and tiling optimizations. To prepare this data for our GNN-based approach, we undertook several preprocessing steps. We cleaned the data by removing any corrupted or incomplete graph instances, ensuring the integrity of our dataset. We decomposed the graphs into their constituent features, transforming opcodes into a numerical format suitable for neural network processing for feature extraction. The configurations and runtime labels were normalized to ensure smooth gradient flow during model training. The dataset was split into training, validation, and testing sets to facilitate model development and evaluation.

Our exploratory data analysis revealed several insights. Runtime distributions varied significantly across different graph structures and configurations, indicating the complexity of predicting runtimes based on graph features alone. Certain operations, particularly those associated with high-dimensional tensor manipulations, showed a strong correlation with increased runtimes. This comprehensive data preparation and exploratory analysis laid the groundwork for the development of our GNN models, allowing us to approach the runtime prediction problem with a robust data-driven methodology.

Proposed Methodology

Following from the idea of representing each model as a graph, we propose to use a GNN (Graph Neural Network) as the base architecture to solve this problem. We will experiment on whether the problem is better suited for graphs. If these experiments yield, as we expect, that a GNN is a superior architecture. Then, we will explore different GNN variants to see which one best performs on our problem. We hypothesize that GNNs will perform best here because of the input's inherent graph-like structure.

Furthermore, most of the industry strategies that employ GNNs use a variant of a GraphSAGE (Hamilton, n.d.). GraphSAGE famously debuted the use of the SageConv layer (short for Sage Convolution), which showed great promise for GNN problems. Specifically, companies like Uber and Pinterest both utilize SageConv in their GNNs because of its inherent ability to scale, but also to develop resilient node embeddings (Hamilton, n.d.). We hypothesize that SageConv will perform the best on our task, since developing resilient node embeddings given the high-range of possibilities for each node feature is very important. For example, nodes can have various configurations (depending on the dimensions of the input) and about 120 different op_codes (which represent the operation they represent such as matrix multiplication, convolution, etc.). Therefore, given that our dataset is only around 250 graphs (with each graph having thousands of different configuration amounting to about 30,000 different training inputs), we need the model to generate a relatively resilient node embedding since the training set will likely not have many instances of certain op_code and configuration pairs.

We will utilize Kaggle and take advantage of the free TPU processing units provided by Google to train our models. We are using Kendall Tau Correlation as a metric to measure the model performance. This metric measures how accurate our model rankings are when compared with real ranking by using concordant and discordant observations. We will then submit our models to Kaggle which will then take care of evaluating our model against the hidden test set and calculate the Kendall Tau Correlation.

Experiments

First, we built a Vanilla GNN and compared it to a simple Neural Network with the same embedding dimensions (32). We then trained both on the same amount of open competition data and then submitted both models to Kaggle. As expected, the neural network did quite poorly, since it wasn't able to utilize the graph-like structure of the problem. This output did not surprise the team, because it simply isn't a problem that a neural network will do very good on. To draw an analogous example, imagine we attempted to image classification or recognition with a neural network? Clearly, we would use a CNN for the aforementioned tasks, so similarly, we would use a GNN for our task at hand.

Model	Kendall Tau Correlation
Vanilla GNN	0.57545
Vanilla NN	0.34754

Now that we are confident that the GNN is a superior architecture to the neural network, we began to try and fine-tune the dimensions. We realized the model performs between when we increase the embedding dimension, so we set it to 64 (which is the maximum value that our TPUs train on given our use limit).

Model	Epochs	Embedding Dimension	Kendall Tau Correlation
Vanilla GNN	10	32	0.57545
Vanilla GNN	10	64	0.59432

Lastly, we implemented the SageConv layers into our model similar to how many industry standard GNNs do. We utilize multiple (4) SageConv layers in our new SageGNN and then apply it on the data at hand. Unsurprisingly, when they are both trained with 64 embedding dimensions, the SageGNN performs better.

Model	Kendall Tau Correlation
Sage GNN	0.65411
Basic GNN	0.59432

Contributions

All the team members researched GNNs, existing AI run time algorithms, and understood the dataset together. Shreya and Anvitha worked on creating the Basic Neural Network and GNN, and Rohit and Harsha worked on creating the Sage Conv GNN. We all split up the work when creating the midterm report, presentation, and final report. For example, for the midterm and final presentation, Shreya focused on the introduction and dataset, Rohit focused on the model, Harsha focused on the metric, and Anvitha focused on next steps.

Conclusion

In conclusion, through the Kaggle competition, we were given numerous graph configurations of AI Models in which a node is an operation and an edge represents the tensor being sent from. We predicted the runtime of AI models using a neural network, a normal graph neural network, and a SageConv GNN. We used the neural network as a baseline, and to take advantage of the graph structure of the dataset, we used the GNNs. Additionally, we used Kendall Tau Correlation, a metric that compares the model rankings of runtimes with the real rankings of runtimes, to describe the performance of each model. We discovered that the SageConv GNN performs the best because the SageConv layers effectively aggregate complex features from a sample of extended neighboring nodes. A limitation for this project is compute power; other teams had more compute power and trained a model that reached a higher score. Some future work that improves Kendall Tau Correlation is to implement cross-attention within the Sage Conv GNN. Ultimately, this work displays predicting AI runtimes is essential to the AI industry today and can be predicted using GNNs.

Source Code Link: https://github.gatech.edu/rarunachalam8/ML4G_Kaggle_Comp

References:

Hamilton, W.L. (no date) *Inductive representation learning on large graphs*, *Stanford*. Available at: https://cs.stanford.edu/people/jure/pubs/graphsage-nips17.pdf (Accessed: 12 December 2023).

Q. Sun et al., "CoGNN: Efficient Scheduling for Concurrent GNN Training on GPUs," SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, 2022, pp. 1-15, doi: 10.1109/SC41404.2022.00044