TPU runtime prediction using GNNs

Valeria Vera Lagos
Georgetown University
vv188@georgetown.edu

Abstract

This paper navigates the challenges of training Deep Learning Models (DLMs) for code generation, particularly in the context of Tensor Processing Units (TPUs) with limited hardware resources. Leveraging a Graph Neural Network (GNN) trained on the TpuGraphs dataset, the study addresses runtime prediction complexities. The GNN model, equipped with residual connections and adjacency matrices for message passing, undergoes evaluation using Ordered Pair Accuracy (OPA) and Listwise Maximum Likelihood Estimation (ListMLE) loss metrics. Despite resource constraints, the model showcases promising performance, achieving a notable 75.01% OPA on the NLP dataset. This research sheds light on the potential of GNNs for advancing computational graph layout optimization, emphasizing the need for further exploration in hyperparameter tuning to optimize efficiency in DLMs and TPUs.

1 Introduction

Deep Learning model (DLM) training is still limited by hardware availability and cost in today's computing environments. For effective code generation, it would be helpful to measure the models' runtime and record the amount of hardware used while they were operating. Unfortunately, even the evaluation of runtime becomes difficult due to the complexity of modern processors like deep learning accelerators (also known as "Tensor Processing Units," or "TPUs") and the complexity of DLMs. In order to solve these issues, this paper trains a graph neural network to create a model that predicts runtime of DLMs on TPUs by leveraging a Google dataset, TpuGraphs [1].

Prior work [2] has employed Graph Neural Networks (GNN) to assess the runtime of two types of optimization compilers methods used in tensors using a similar dataset. Several applications, including machine learning models and optimization algorithms, rely on graphs as their fundamental building bloc. Given the nature of the data a GNN has been implemented.

The subsequent sections go deeper into the details of the GNN model architecture, the characteristics of the dataset, the experimental setup, and the outcomes of our evaluations. Through this exploration, we seek to provide insights into the potential of GNNs in advancing computational graph layout optimization and shaping the future of graph-based applications.

2 Data

A tensor program can be represented as an acyclic and directed graph. A node in a graph represents a tensor operation. Each graph contains a set of nodes a set of edges and context. The graph can process one or more tensors operations into a single output, and an edge connects an output tensor from one node to an input tensor of another node. Meanwhile context is the information related to the general model.

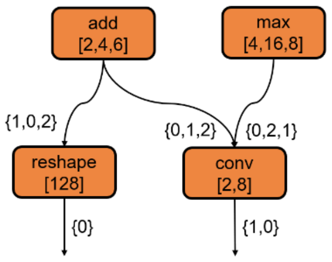

Figure 1. Example of a Graph

Each row of the dataset contains a graph tensor and its corresponding context which is a run time tensor expressed in seconds. Two distinct compiler optimizations separate the dataset. A compiler optimization is a process that looks to reduce the model run time. The first compiler optimization is “layout” where tensors are arranged in memory based on various feats like size and shape. The second one is “tile” which processes tensors by dividing them into smaller “tiles”. Tile optimization is sourced from an XLA (accelerated linear algebra) model meanwhile layout optimization is sourced from both XLA and NLP (natural language processing) models. Given that the layout method tensors can be arranged the dataset is subdivided into 2 different search strategies: the random split method, which involved randomly partitioning programs into sets, and the manual split method, which involved manually selecting the test set to minimize the subjective similarity of programs between the training and other two sets. The characteristics of the models such as batch sizes, including the number of graphs, sampled nodes per graph, and configurations per graph, are used as hyperparameters during testing.

Layout Dataset	*Search strategy*	*Test rows*	*Train rows*	*Valid rows*
XLA	Default	8	61	7
	Random	8	69	7
NLP	Default	17	179	20
	Random	17	77	20

Table 1. Layout Dataset content

TILES DATASET	*Test rows*	*Train rows*	*Valid rows*
XLA	832	5716	676

Table 2. Tiles Dataset content

3 Model Design

Graph Neural Networks (GNNs) are a class of neural network architectures specifically designed to handle graph-structured data, where entities (nodes) and relationships (edges) form a complex network. This work utilizes a Neural Network-based architecture with residual connections tailored for the task of configuration layout optimization, implemented in TensorFlow. The model input is a custom embedding layer to represent operation nodes in the graph. Furthermore, the embedding goes through different number of graph convolution layers and incorporates a multi-layer perceptron (MLP) for node-level feature processing on each layer. Each MLP has the same activation function (Leaky Relu), regularization (l2reg), and a bias term. Later, the architecture introduces residual connections to enhance information flow through all layers; It incorporates it by adding the input features to the output of each layer. Additionally, the model utilizes adjacency matrices for various edge sets, enabling message-passing operations, which involves the propagation of information between nodes in a graph.

During the training process, the model incorporates a Listwise Maximum Likelihood Estimation (ListMLE) loss function, and it is optimized using the Adam optimizer. Finally, to prevent overfitting, the architecture employs segment dropout, where edges are randomly sampled during training and it includes early stopping based on the validation metric, ensuring efficient convergence, and preventing unnecessary computation.

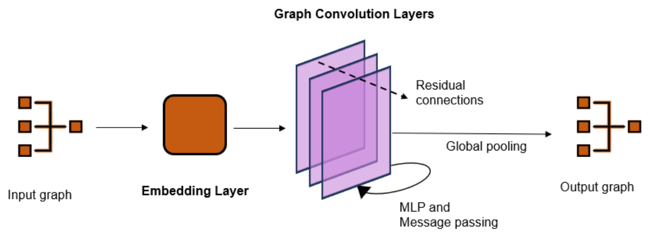

Figure 2. GNN Architecture description

4 Model Evaluation

The experiments where run on a google-collab environment utilizing a TPU engine with 12.7 GB of RAM and 107.7 GB of Memory.

The model is evaluated using the Ordered pair accuracy (OPA) metric and the ListMLE loss on the entire output graph, not only the context (run time). The OPA metric used in graph-related tasks since it is suitable for scenarios where the order of predictions matters, in this scenario it is calculated by comparing the predicted rankings with the actual rankings of configurations.

The model's performance is evaluated on a variety of datasets, search strategies, and hyperparameters that regulate the amount of RAM used for training: maximum number of nodes to retain during training used in dropout, maximum number of configurations: features and target value per graph during training, and batch size to control the number of configurations processed at a time when running inference on the test set.

5 Results

Multiple experiments were performed, unfortunately the resources from the environment were not sufficient to train and test the model efficiently. The Layout dataset models took between 70 to 350 minutes to run for the different datasets. Meanwhile, the Tiles dataset model was not able train without crashing due to the low RAM. Different hyperparameters were explored to reduced the RAM memory usage. The next results were achieved with the next hyperparameters: batch size equal to 10, configurations per graph equal to 2 and maximum number of nodes during dropout equal to 100.

Table 3 reports the total ListMLE and OPA percentage results across all models trained with different datasets and same hyperparameters. The total best OPA metric of 75.01% evaluating with the testing dataset was achieved with the NLP dataset using a default search strategy. The default search strategy could have a better OPA during testing given that the datapoints were sampled for the model to have a better generalization of our data, therefore it was not overfitted, in comparison to the random strategies that performed worse for this specific dataset.

We can observe how the model produced a greater OPA with the XLA dataset using the validation dataset. The expected behavior would be a higher OPA during training rather than testing but, as previously mentioned, the results are impacted by the small amount of data rows we have on each set.

Dataset	*Search Strategy*	*Training*		*Validation*		*Testing*	
		ListMLE	OPA	ListMLE	OPA	ListMLE	OPA
XLA	Default	.720	50.82	.715	42.86	.790	36.52
	Random	.664	62.32	.597	**85.71**	.706	55.53
NLP	Default	.701	56.52	.668	70	.646	**75.01**
	Random	.673	**64.25**	.695	60	.717	52.46

Table 3. Evaluation of the model in terms of total ListMLE and OPA percentage

6 Conclusion

This study, presented a comprehensive approach to computational graph layout optimization, leveraging a Graph Neural Network (GNN). The exploration began with a detailed examination of the underlying data, comprising diverse computational graph layouts sourced from domains such as "xla" and "nlp." An architecture, designed for layout optimization tasks was provided. This GNN integrates a multi-layer perceptron (MLP) with residual connections, enabling efficient message-passing and feature processing across nodes and edges. The model's implementation showcased its flexibility in handling different graph structures and its ability to adapt to varying optimization scenarios.

The model is evaluated using Ordered Pair Accuracy (OPA) and ListMLE loss metrics across various datasets, search strategies, and hyperparameters. Despite resource limitations, the model demonstrates promising performance, achieving a notable OPA of 75.01% on the NLP dataset with a default search strategy. However, challenges such as training time and RAM constraints necessitate further exploration of hyperparameter settings to optimize model efficiency. This work contributes valuable insights into the potential of GNNs for advancing computational graph layout optimization and shaping the future of graph-based applications in the context of DLMs and TPUs.

In conclusion, our study contributes to the evolving landscape of computational graph optimization by introducing a versatile GNN architecture and conducting thorough experiments on real-world datasets. As future work it would be interesting to evaluate the designed model with. As computational graphs continue to play a pivotal role in various domains, our work opens new possibilities for enhancing efficiency and performance in critical graph-based applications.

References

[1] Mangpo Phothilimthana, Sami Abu-El-Haija, Bryan Perozzi, Walter Reade, Ashley Chow. (2023). Google - Fast or Slow? Predict AI Model Runtime. Kaggle. https://kaggle.com/competitions/predict-ai-model-runtime

[2] Kaufman, S., Phothilimthana, P., Zhou, Y., Mendis, C., Roy, S., Sabne, A., & Burrows, M. (2021). A learned performance model for tensor processing units. Proceedings of Machine Learning and Systems, 3, 387-400.