Prediction of the Resource Consumption of Distributed Deep Learning Systems

Authors: Gyeongsik Yang, Changyong Shin, Jeunghwan Lee, Yeonho Yoo, and Chuck Yoo
Department of Computer Science and Engineering, Korea University, South Korea

Presenter: Dipak Acharya
HPC Seminar in Dept. of CSE at UNT on 10/12/2023

Gyeongsik Yang, Changyong Shin, Jeunghwan Lee, Yeonho Yoo, and Chuck Yoo. 2022. Prediction of the Resource Consumption of Distributed Deep Learning Systems. Proc. ACM Meas. Anal. Comput. Syst. 6, 2, Article 29 (June 2022), 25 pages. https://doi.org/10.1145/3530895

Background: Distributed Training (DT)

- Gradients calculated by several workers is aggregated and applied to the entire model
- Parameter server (PS) and all reduce
- 2 axes of complexities

01 **Training Settings**

- GPUs
- Parameter Servers
- Same server GPUs or PCIe connected
- 40GbE or High speed RDMA

02 **WorkLoads**

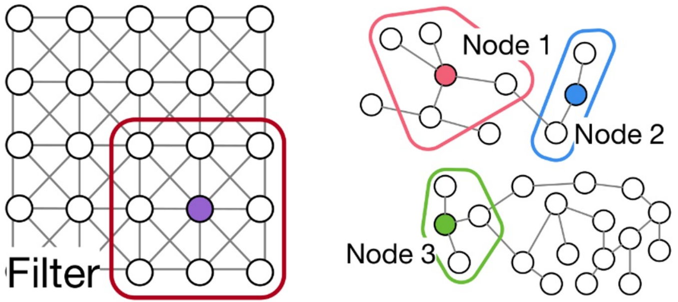
- Models
- Datasets
- Hyperparameters

Background: Graph Neural Networks (GNN)

- Node can have variable number of connections
- Difficulty in neural network design

Solution:

- Convert each node to fixed sized vectors (embeddings) using graph layers
- Node embeddings aggregate information of neighbors within certain hops
- By stacking *n* layers, we can aggregate information from *n-hop* neighbors
- This *n* value is a GNN specific hyperparameter
- To accommodate different node sizes, a fixed size vector (embedding) is created for graph
- Use embeddings vector on traditional machine learning algorithm such as MLP

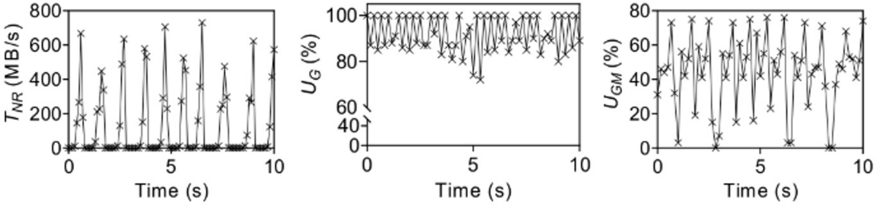


(a) CNN on image. (b) GNN on graph.

Motivating Example

- **Workload 1**: NMT_Medium [40], Europarl dataset [41] (batch size 32, asynchronous training)
- **Workload 2**: DenseNet40_k12 [34], CIFAR-10 dataset [42] (batch size 512, synchronous training)
- **Workload 3**: Inception v3 [58], ImageNet dataset [24] (batch size 128, asynchronous training)

- Profiling resource consumption on 3 DT workloads
- Each resource shows cyclic pattern of high and low consumption
- 4 key prediction resources were selected with 3 metrics (burst duration, idle duration and burst amount) for each
  - *GPU utilization (U_G)*
  - *GPU Memory Utilization (U_{GM})*
  - *Network Tx Throughput (T_{NT})*
  - *Network Rx Throughput (T_{NR})*



(a) T_{NR}, workload 1. (b) U_G, workload 2. (c) U_{GM}, workload 3.

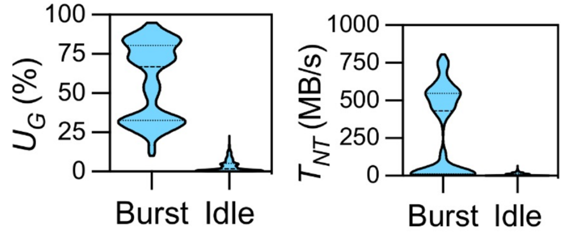
	T_{NR} **(workload 1)**	U_G **(workload 2)**	U_{GM} **(workload 3)**
Burst	374.3 MB/s, 0.33 s	98.8%, 0.19 s	54.4%, 0.64 s
Idle	4.5 MB/s, 0.7 s	84.6%, 0.21 s	16.8%, 0.28 s

Motivation: Sensitivity to DT workloads

Pattern	U_G	U_{GM}	T_{NT}	T_{NR}
I	7.77%	9.26%	0%	0%
BIB	92.02%	90.63%	100%	100%
B	0.21%	0.11%	0%	0%

Workload = {model, dataset, hyperparameters}

Use clustering on profiled data to classify burst and idle data points



(a) U_G. (b) T_{NT}.

Most resources follow primarily BIB cyclic pattern (B: burst, I: idle)

Burst amount varies highly, Very sensitive to DT workloads

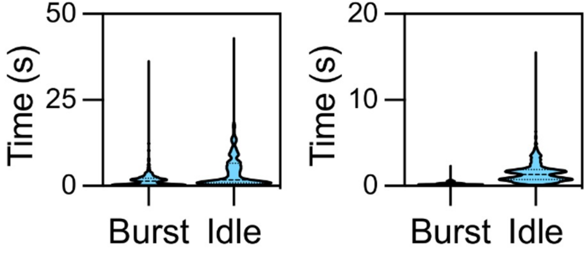
Idle amount varies relatively less

Both Burst duration and Idle duration have a high range for U_G

Burst duration is 88% shorter than Idle duration for T_{NT}

Burst duration is very sensitive for U_G and U_{GM}

Idle duration for sensitive for all types of resources



(a) U_G. (b) T_{NT}.

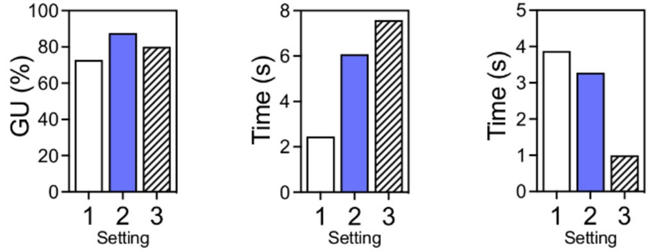
Motivation: Sensitivity to DT Settings

Settings = {GPUs, number of PS, workers, network Interconnections}

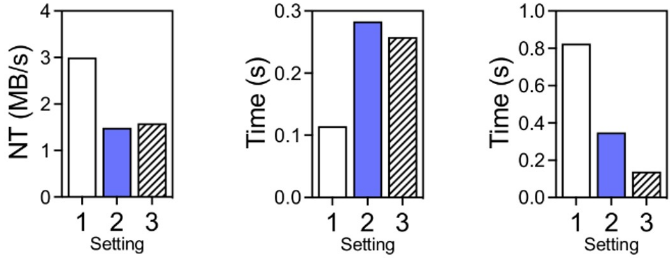
Three Settings tested on same workload (ResNet56 + CIFAR10, bs=512)

- V100 - P1W2 / ho-PCIe
- 2080Ti - P2W2 / he-40G
- Titan RTX - P2W2 / he-40G

Results show that resources are highly sensitivity to DT Settings



(a) burst amount. (b) burst duration. (c) idle duration.



(a) burst amount. (b) burst duration. (c) idle duration.

Related Work

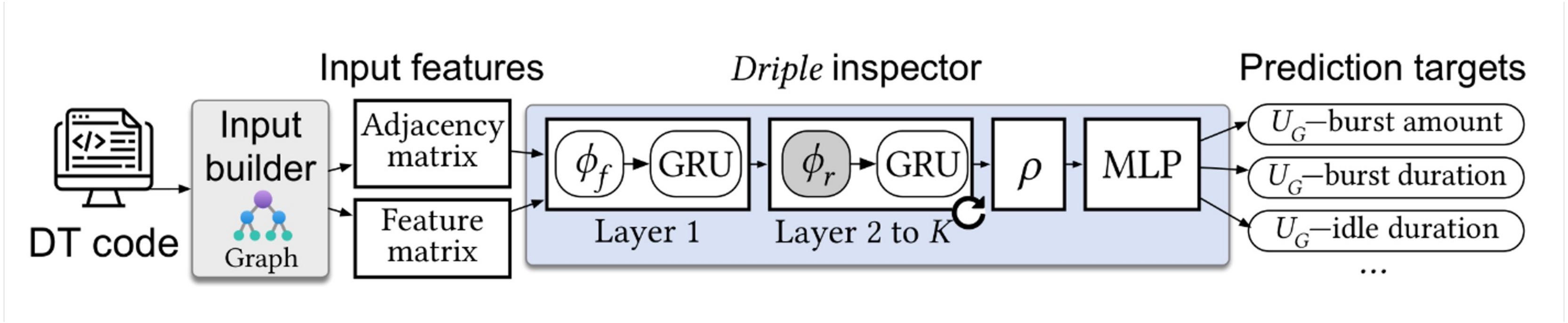
	Paleo [54]	Justus et al. [36]	Daydream [73]	Habitat [27]	*Driple*
Inspection metric	Computation time per iteration, communication time per iteration	Training time per batch	Training time per iteration	Training time per iteration	Burst amount, burst duration, idle duration of four resources.
Inspection method	Mathematical modeling	Neural network	Simulation on graph	MLP	GNN, TL
Inspection (input) scope	Two kinds of convolution layer (matrix multiplication and fast Fourier Transform)	Convolution, fc	Optimization on models (e.g., FusedAdam optimizer)	2D-convolution, LSTM matrix multiplication, linear	Any operations expressed by graph
Distributed training	No detailed models	No	All-reduce	No	Yes (data parallel, PS)
ML library	TensorFlow	TensorFlow	PyTorch	PyTorch	TensorFlow

Existing works:

Heavy focus on overall training time or iteration time, ignoring the actual resource utilization

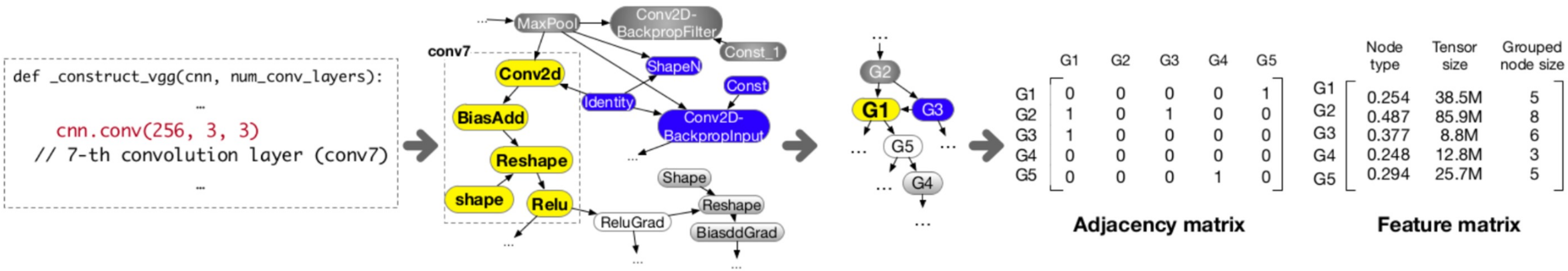
Use of fixed size mathematical modeling, simulation or ML; limited prediction scope

Driple Design



- Input Builder
- *Driple* Inspector

Input Builder



(a) DT workload code.

(b) computational graph. (461 nodes and 606 edges)

(c) grouping. (100 nodes and 203 edges)

(d) input features.

Notations: Graph = $G(N, E)$, $n_i \epsilon (N)$, X_n is the features of node n, $N(n_i)$ is the immediate neighbourhood of node n_i

- Each layer converted into low level operations, where operations become *n* in the Graph
  - Eg. ADD, MATMUL
- Other components such as hyperparameters, constants, input data all become either node or edges of the graph
- Edges from n_i to n_j = e_{ij} => Gives the execution order

Input Builder: Grouping

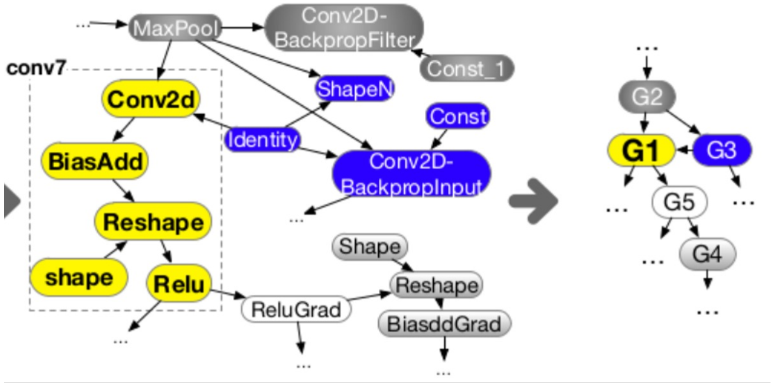
A graph with *m* nodes is reduced into a graph of size *M*

- Reduces the size of $|N|$ and $|E|$;
- Enables Batching (Batching requires all the graph in a batch having same number of nodes)

Uniform Grouping :
- Same *M* for all batches

Proportional Grouping:
- Sort the graphs based on m, then group by batch size
- $M = log_{10}v$; v = average number of nodes in a batch

Grouping is done based on *fluid Communities algorithm.* Randomly selects *M* seed nodes and group remaining nodes around those



Input Builder (Cont.)

Features selected for X_n are:

- Node type (frequency encoding)
- Tensor Size
- Grouped node size

$$
\begin{array}{c} \\ G1 \\ G2 \\ G3 \\ G4 \\ G5 \end{array}
\begin{array}{c} \begin{array}{ccccc} G1 & G2 & G3 & G4 & G5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}
$$

Adjacency matrix

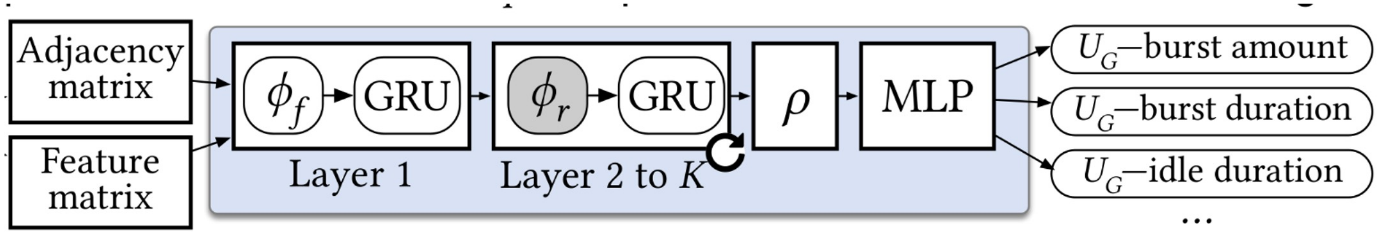
$$
\begin{array}{c} \\ G1 \\ G2 \\ G3 \\ G4 \\ G5 \end{array}
\begin{array}{c} \begin{array}{ccc} \text{Node type} & \text{Tensor size} & \text{Grouped node size} \end{array} \\ \begin{bmatrix} 0.254 & 38.5M & 5 \\ 0.487 & 85.9M & 8 \\ 0.377 & 8.8M & 6 \\ 0.248 & 12.8M & 3 \\ 0.294 & 25.7M & 5 \end{bmatrix} \end{array}
$$

Feature matrix

Input features

- **Adjacency Matrix:**
  - m x m matrix, $m = |N|$
- **Feature Matrix:**
  - m x f matrix, $f = |X_n|$
  - X_n contains features, such as node type and tensor size.

Driple Inspector



- Graph Layers perform update function(ϕ)
    - First layer: ϕ_f
    - Next Layers: ϕ_r (Same weight/bias)
    - ϕ_r layers are repeated K times; K = m/2
- Node embedding of node n_i for layer k is

$$h_{n_i}^{k}$$

- Graph Convolutional Network (GCN), Uses normalized mean for aggregation update
- Gated Recurrent Units (GRU) used after each graph layer, Reduces over-smoothing problem in deep neural networks

Driple Inspector (Cont.)

$$h^1_{n_i} = \phi_f = \mathtt{AGGREGATE}^1(X_{n_j} | n_j \in \mathcal{N}(n_i)), i = 1, \ldots, m$$

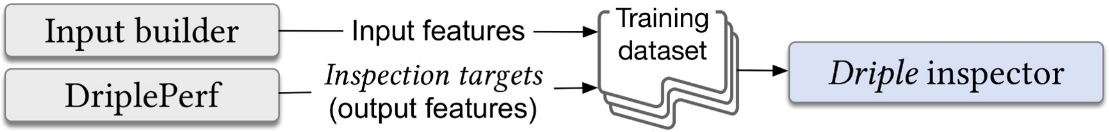
$$h^k_{n_i} = \phi_r = \mathtt{COMBINE}^k(h^{k-1}_{n_i}, \mathtt{AGGREGATE}^k(h^{k-1}_{n_j} | n_j \in \mathcal{N}(n_i))), i = 1, \ldots, m$$

$$h_{G_l} = \rho = \mathtt{POOL}(h^K_{n_i} | n_i \in N_l)$$

- Graph Embedding (*h_{Gl}*) for l^{th} graph in training data is produced by graph readout layer (ρ) after pooling
    - Set2set is used for pooling (Uses LSTM Neural Network)

- MLP used on graph embeddings produced by ρ
    - 3 Fully Connected Layers
    - 12 Prediction targets for resources consumption metrics

Training of *Driple* Inspector



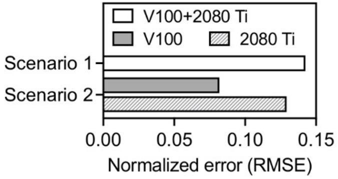
- Input builder generates the input features (graphs) for the inspector
- DriplePerf generates the prediction targets for the data
- Combining these two we can build the training data for Driple

DriplePerf: Measures the resource consumption by executing the DT code. Performs K means clustering to divide the data points into burst and idle points

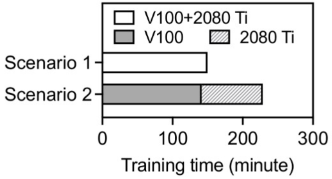
The inspector is trained multiple iterations on the data produced by this way

Transfer Learning (TL)

- Common Inspector for different settings: low accuracy
- Different inspector for each setting: High training time
- Solution: **Transfer Learning**



(a) prediction error.

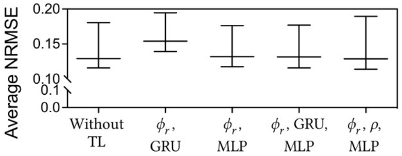


(b) training time.

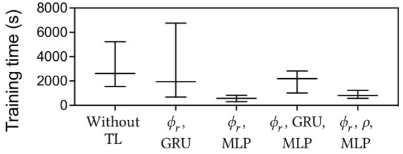
Fine-Tuning TL:

(ϕ_r ,GRU) has lower accuracy compared to others, whereas (ϕ_r ,MLP) has lowest training time

So, (ϕ_r ,MLP) was selected for fine tuning



(a) prediction error.

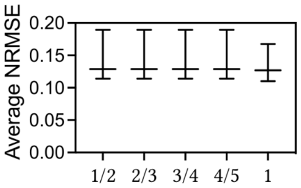


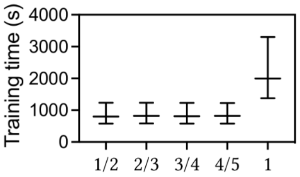
(b) training time.

ϕ_r Partitioning:

ϕ_r Layers needs to be partitioned in layers that use pre trained inspector parameters and layers that update parameters on TL

By testing different ratios, it was selected to be ½





Evaluation: Design Choices of *Driple*

Metric Used: Normalized Root Mean Square Error (NRMSE) (average of 12 prediction targets)

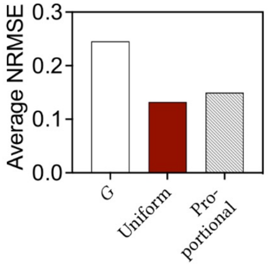
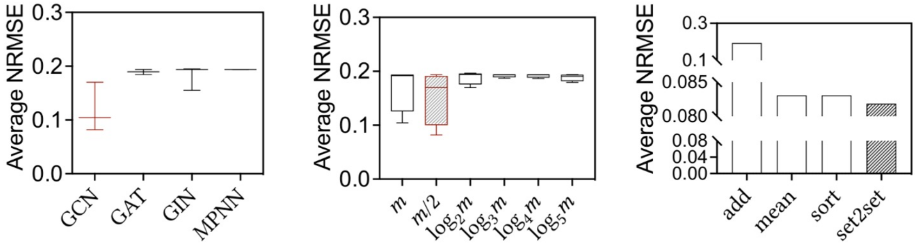


Fig. 13. Grouping policies



(a) GNN algorithms. (b) number of layers. (c) ρ.

Input builder(grouping): G(no grouping) has highest error, proportional has 46% lower, Uniform has 12 % lower

Inspector (GNN algorithm): GCN has the lowest average NRMSE

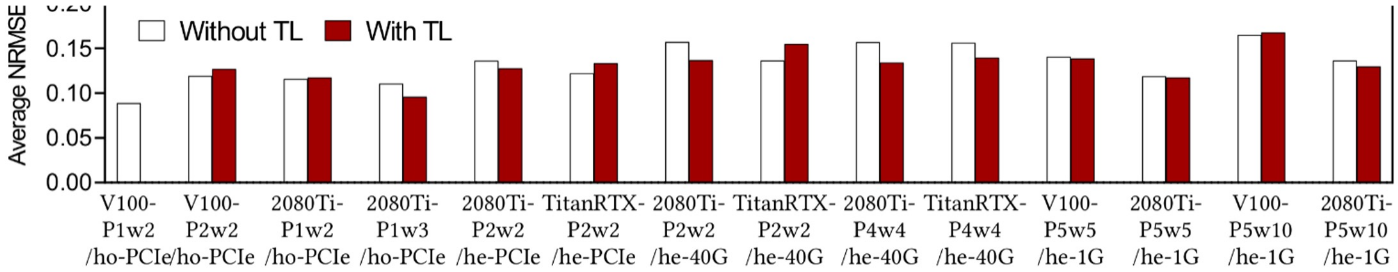
Inspector (K): Lowest average NRMSE given by *m/2*

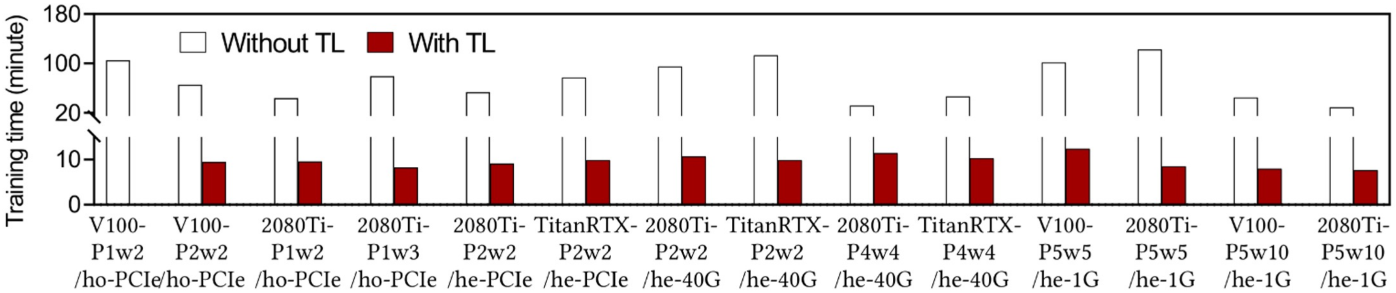
Inspector (Readout Layer, ρ): Lowest average NRMSE given by *set2set*

Evaluation: TL Effectiveness

TL on Inspector for V100-P1w2/ho-PCIe (Selected because it shows highest accuracy)

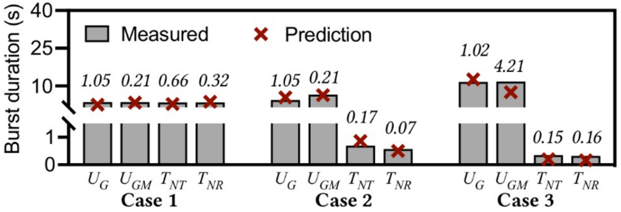
Average NRMSE is mostly similar or lower, Training time is highly reduced (by 7.3X on average)



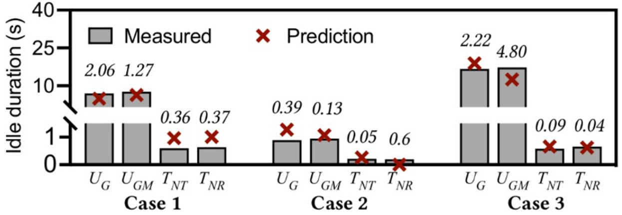


Evaluation: Prediction of Resource Consumption

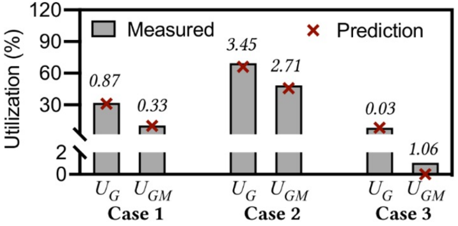
- **Case 1**: ResNet44 model, CIFAR-10 dataset, V100-P2w2/ho-PCIe (asynchronous training)
- **Case 2**: GoogLeNet model, ImageNet dataset, TitanRTX-P2w2/he-40G (synchronous training)
- **Case 3**: Transformer-AAN, Europarl dataset, V100-P2w2/ho-PCIe (asynchronous training)



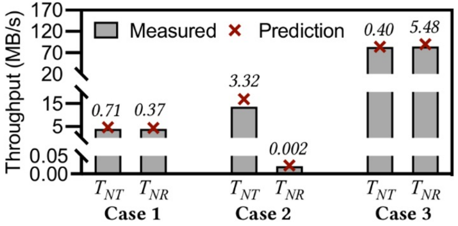
(a) burst duration.



(b) idle duration.



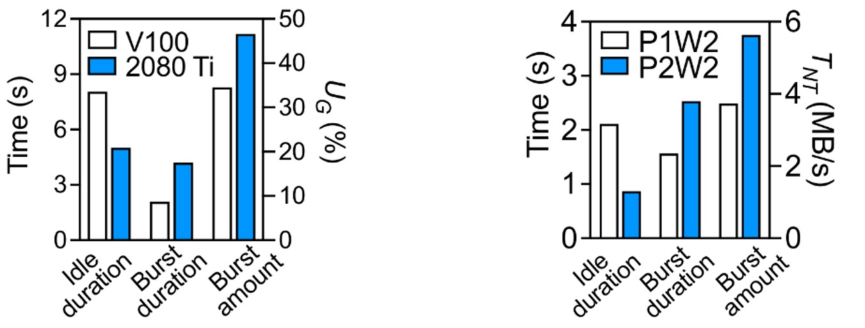
(c) burst amount of U_G and U_{GM}.



(d) burst amount of T_{NT} and T_{NR}.

On average, the percentage errors are 11%, 9%, 17%, and 15%, for U_G , U_{GM} , T_{NT} , and T_{NR} , respectively.

Evaluation: Applications of *Driple*



(a) GPU change. (b) PSs and workers change. (c) batch size change.

A. 2080Ti has higher burst duration and lower idle duration, which means better U_G compared to V100
B. Increasing the parameter Server from 1 to 2 Increases burst duration and decreases idle duration, overall execution time stays the same
C. Observations for batch size change
   a. Large Batch sizes can be handled
   b. Increasing batch sizes increases execution time by lower factor, which implies using large batch size can reduce overall training time

Contributions

- GNN for predicting resource consumption for deep learning using DT on variety of workloads and settings
- Use 4 Key resources for prediction; For each resource, ***Driple*** predicts 3 metrics, burst amount, burst duration and idle duration
  - *GPU utilization* (U_G)
  - *GPU Memory Utilization* (U_{GM})
  - *Network Tx Throughput* (T_{NT})
  - *Network Rx Throughput* (T_{NR})
- *Driple* can achieve good prediction accuracy for resource consumption on variety of DT workloads and settings
- Use of Transfer learning reduces the required dataset by up to 2.5X and training time by 7.3X while maintaining prediction accuracy of a model without transfer learning

Discussion

- Use *Driple* on other DT strategies such as all reduce
- *Driple* uses Tensorflow API for graph extraction, other libraries also use graph to store the model
- Including resources of PS and workers other than U_G, U_{GM}, T_{NT} and T_{NR}
- Node features to include
- Reduce time for dataset generation
- Boundary of TL