ETS: Deep Learning Training Iteration Time Prediction based on Execution Trace Sliding Window

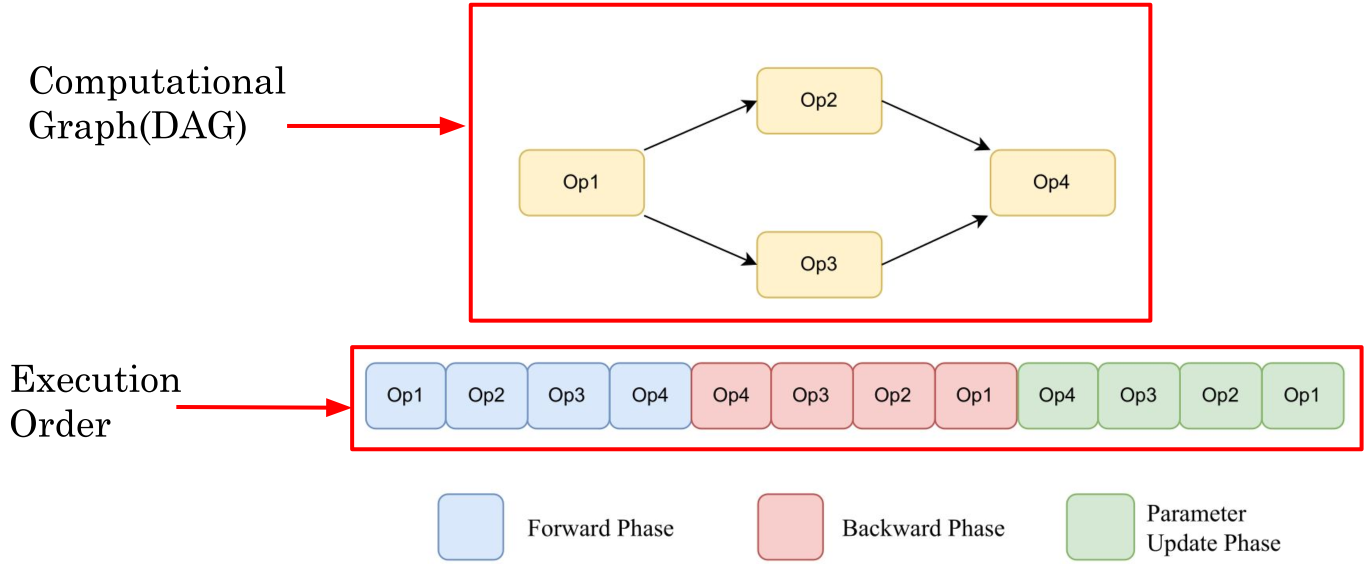
Zichao Yang, Hao Guo, University of Chinese Academy of Sciences; Heng Wu, Yuewen Wu Hua Zhong, Wenbo Zhang, Institute of Software,Chinese Academy of Sciences ; Chuan Zhou, Minzu University of China ; Yan Liu, Inspur software Co., Ltd.

https://dl.acm.org/doi/10.1145/3625549.3658658

Presenter

Dipak Acharya
University of North Texas
Denton TX

Deep Learning(DL) Training



DL Latency prediction

Operator Based Prediction

- DL models treated as set of operators, predict latency of each operator and combine the result
- These models can be analytical or data driven
- Eg. PALEO, Habitat

- They cannot capture the order of operator and the DL framework overhead resulting in low accuracy.

Graph Based Prediction

- DL models treated as DAGs, predict the latency of entire model
- Most methods train GNNs to predict the end to end latency
- Eg. Dnnperf, Driple, BRP-NAS

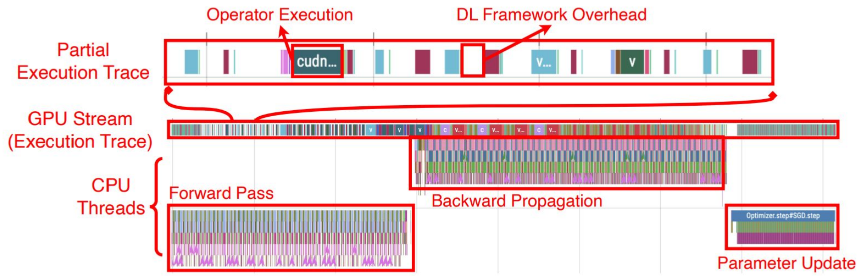
- These methods need to build large models with huge numbers of parameters and often suffer from slow convergence

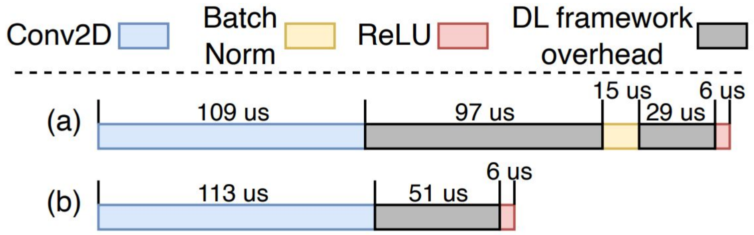
Execution Trace

- Profiled operator execution order and time in the GPU stream

- Operators’ Execution is Sequential with framework overhead for each Operator. Execution order remains same across iterations.

- The execution order of the operators has significant impact on the DL framework overhead





DL model Diversity

- DL model sizes range from few operators such as LeNet (6 operators), SqueezeNet (66 operators) to extremely large number of operators such as Transformers (1201 operators) and Swin (1133 operators).

- Diverse size of DL model require a comprehensive predictor for all model sizes results in slower convergence.

- A leading operator may not exert significant overhead on a distant trailing operator; This is because they are not executed in parallel and lack any direct dependencies.

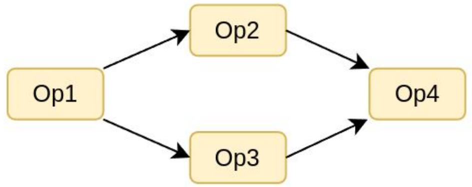
- This allows the DL execution trace to be treated as a collection of multiple smaller slices which effectively reduces the overall feature size for faster convergence.

Challenges

- How to incorporate DL framework overhead into the prediction model?
- How to enable fast convergence for predictor with diverse model sizes?

Latency Prediction Problem

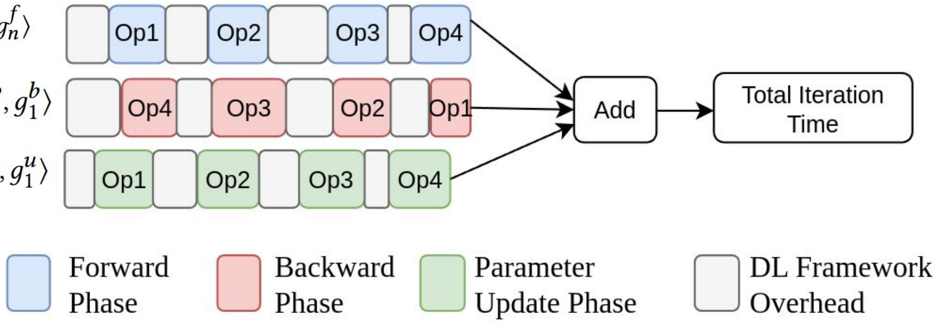
$$M = (op_1, op_2, \ldots, op_n)$$



$$T^f(M) = \langle e_1^f, g_1^f, e_2^f, g_2^f, \ldots, g_{n-1}^f, e_n^f, g_n^f \rangle$$

$$T^b(M) = \langle e_n^b, g_n^b, e_{n-1}^b, g_{n-1}^b, \ldots, e_1^b, g_1^b \rangle$$

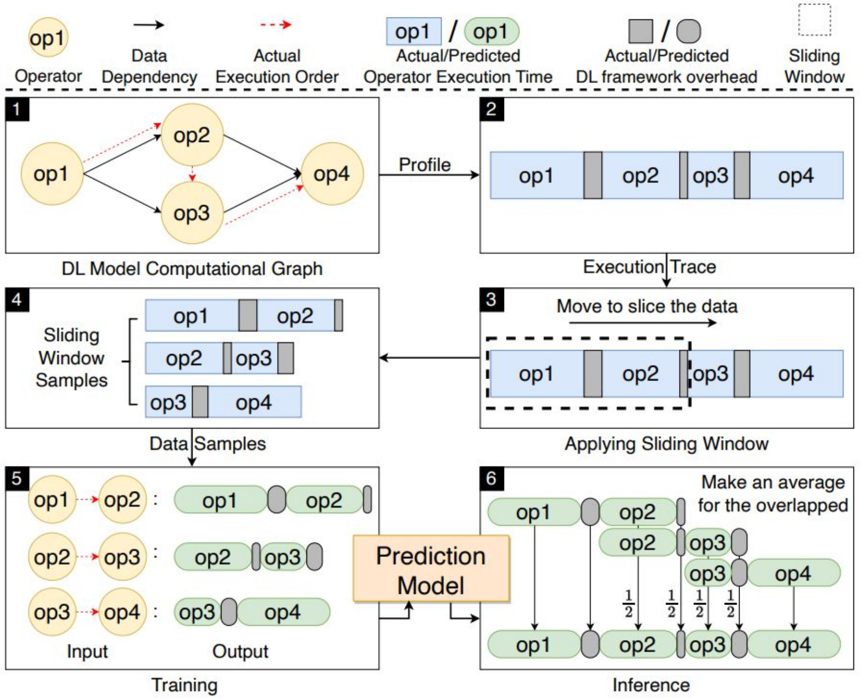
$$T^u(M) = \langle e_n^u, g_n^u, e_{n-1}^u, g_{n-1}^u, \ldots, e_1^u, g_1^u \rangle$$



$$T(M) = T^f(M) + T^b(M) + T^u(M)$$

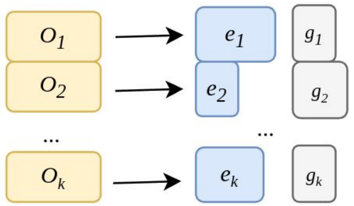
ETS: Execution Trace Sliding Window

1. Submit DL model with 4 operators
2. Profile model to get execution trace with operator execution time and DL framework overhead
3. Apply the Sliding window on the execution trace
4. Get the sliding window samples
5. Use the sliding window samples to train a prediction model
6. Use the trained sliding window to infer the execution trace for every sliding window. Overlapped parts are averaged



Prediction Model

Model Input/Output



$$\begin{bmatrix} o_i^{f/b/u} \\ o_j^{f/b/u} \\ \cdots \\ o_k^{f/b/u} \end{bmatrix} \rightarrow \begin{bmatrix} e_i^{f/b/u} & g_i^{f/b/u} \\ e_j^{f/b/u} & g_j^{f/b/u} \\ \cdots & \cdots \\ e_k^{f/b/u} & g_k^{f/b/u} \end{bmatrix}$$

Operator Features

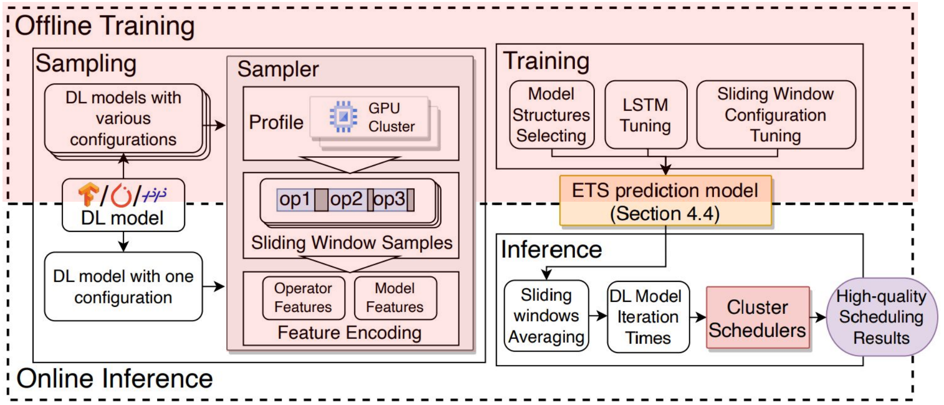
Category	Name	Description
Operator(o^t)	Operator type	Eg. Conv2D, ReLU, AvgPool2D
Phase(o^p)	Training Phase	Forward, backward or Parameter update
Hyperparameter(o^h)	Hyperparameters	Eg. Number of channels, kernel size
Tensor(o^d)	Tensor Size	Size of input, output and temporary tensor
Computational Cost(o^c)	FLOPS	Number of Floating point operations
Batch Size(o^s)	Batch Size	Training batch size, log of 2 scale
Optimizer(o^m)	Optimizer parameters	Eg. Optimizer type, learning rate

$$o = P\,(\,o^t \mid o^p \mid o^h \mid o^d \mid o^c \mid o^s \mid o^m)$$

System Workflow

Offline Training:

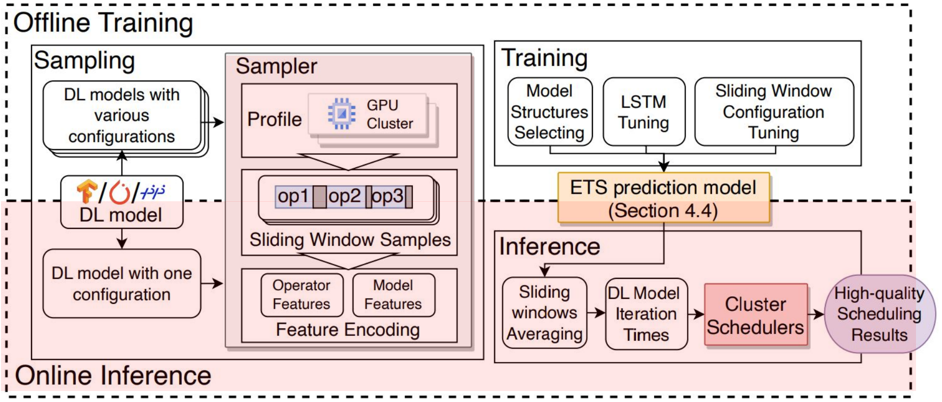
- Create a diverse range of DL configurations to create a training dataset
- Model configs are generated by varying the hyperparameters
- Create sliding window samples and feature encoding
- Select optimal model structure, tune Sliding window configuration and Train the prediction model



System Workflow

Online Inference:

- Make prediction on unseen DL models
- Only one configuration needs profiling for each DL model to determine the operators' execution order
- A single DL model only takes around 3 seconds for profiling
- Modern schedulers allow pre running a process for job to detect issues, which can be used to get the execution trace



ETS Settings

Models Evaluated: RNN, LSTM, GRU, transformers, GCN

LSTM was selected as it performed best from these models

LSTM Hyperparameters:

Layers: 4 bidirectional layers	*Optimizer*: Adam ($\alpha = 0.9, \beta=0.999$)
Batch Size: 128	*Epochs*: 100

Sliding Window parameters:

Width: 12	*Stride*: 4

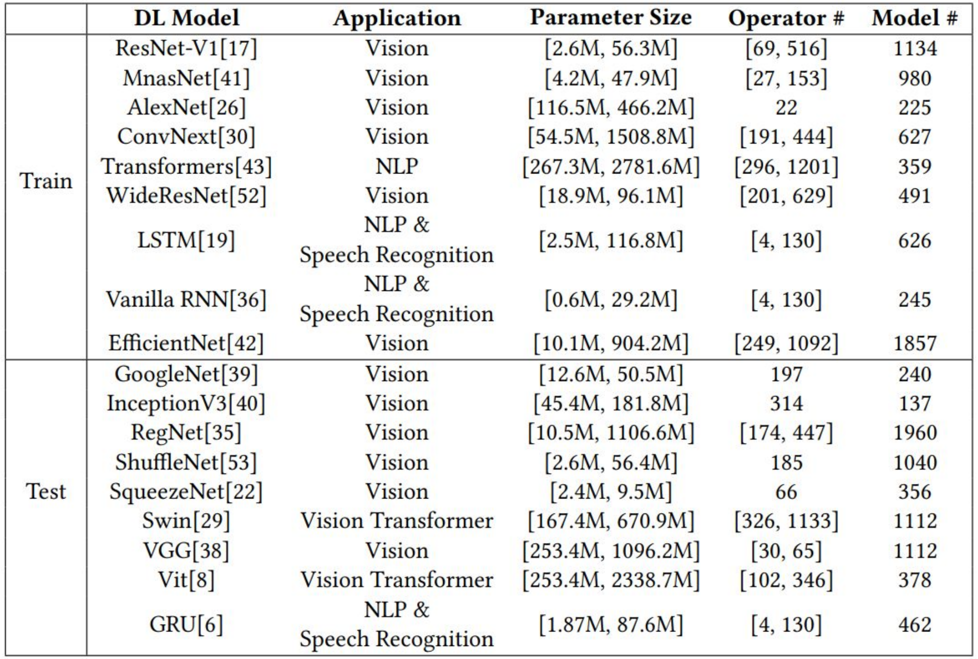
Evaluation: Dataset

HPO Dataset

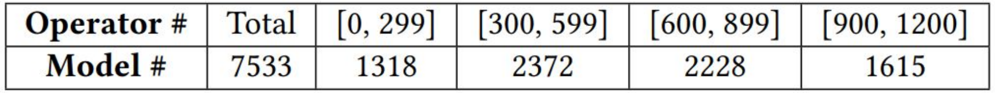
- For each model varying the Hyperparameters to generate diverse model configurations
- Hyperparameters include batch size, input channels, input height, layers, feature size etc

RandWire Dataset

- Using RandWire Neural Architecture Search(NAS) for generating model



	DL Model	Application	Parameter Size	Operator #	Model #
Train	ResNet-V1[17]	Vision	[2.6M, 56.3M]	[69, 516]	1134
	MnasNet[41]	Vision	[4.2M, 47.9M]	[27, 153]	980
	AlexNet[26]	Vision	[116.5M, 466.2M]	22	225
	ConvNext[30]	Vision	[54.5M, 1508.8M]	[191, 444]	627
	Transformers[43]	NLP	[267.3M, 2781.6M]	[296, 1201]	359
	WideResNet[52]	Vision	[18.9M, 96.1M]	[201, 629]	491
	LSTM[19]	NLP & Speech Recognition	[2.5M, 116.8M]	[4, 130]	626
	Vanilla RNN[36]	NLP & Speech Recognition	[0.6M, 29.2M]	[4, 130]	245
	EfficientNet[42]	Vision	[10.1M, 904.2M]	[249, 1092]	1857
Test	GoogleNet[39]	Vision	[12.6M, 50.5M]	197	240
	InceptionV3[40]	Vision	[45.4M, 181.8M]	314	137
	RegNet[35]	Vision	[10.5M, 1106.6M]	[174, 447]	1960
	ShuffleNet[53]	Vision	[2.6M, 56.4M]	185	1040
	SqueezeNet[22]	Vision	[2.4M, 9.5M]	66	356
	Swin[29]	Vision Transformer	[167.4M, 670.9M]	[326, 1133]	1112
	VGG[38]	Vision	[253.4M, 1096.2M]	[30, 65]	1112
	Vit[8]	Vision Transformer	[253.4M, 2338.7M]	[102, 346]	378
	GRU[6]	NLP & Speech Recognition	[1.87M, 87.6M]	[4, 130]	462



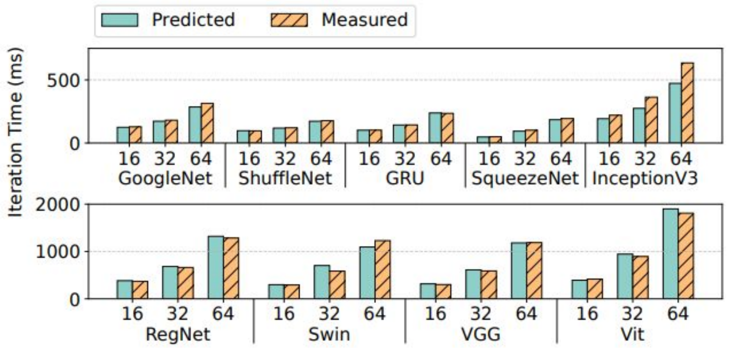
Operator #	Total	[0, 299]	[300, 599]	[600, 899]	[900, 1200]
Model #	7533	1318	2372	2228	1615

Evaluation: Prediction Accuracy

ETS achieves **5.9%** MRE and RMSE of **58.6ms**

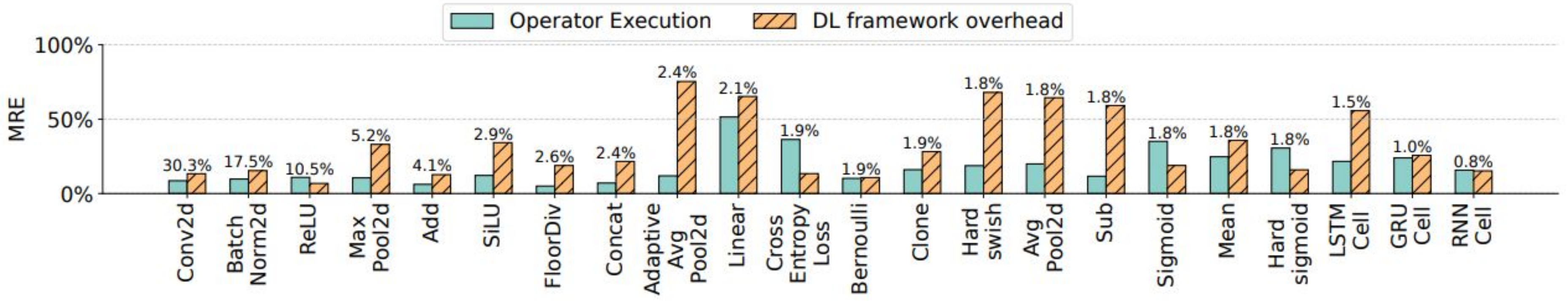
ETS performs better compared to baselines in both HPO and RandWire dataset

Dataset	Metrics	ETS	Habitat	DNNPerf
Overall	MRE(%)	**5.9**	11.9	9.7
	RMSE(ms)	**58.6**	113.6	93.0
HPO	MRE(%)	**5.4**	10.7	7.5
	RMSE(ms)	**41.7**	83.1	62.6
RandWire	MRE(%)	**6.8**	12.5	12.7
	RMSE(ms)	**72.5**	121.8	138.6



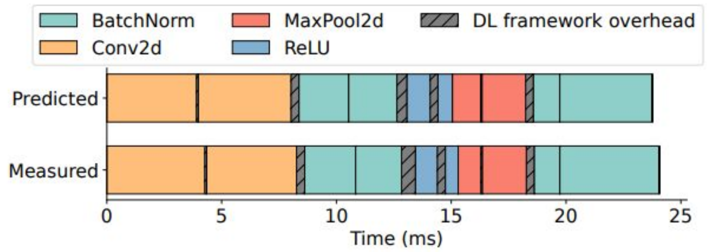
[RandWire] Operator #	Metrics	ETS	Habitat	DNNPerf
Overall	MRE(%)	**6.8**	12.5	12.7
	RMSE(ms)	**72.5**	121.8	138.6
[0, 299]	MRE(%)	**6.7**	10.1	9.2
	RMSE(ms)	**63.6**	105.9	89.2
[300, 599]	MRE(%)	**7.9**	13.6	10.2
	RMSE(ms)	**80.6**	135.3	91.0
[600, 899]	MRE(%)	**6.5**	11.5	13.7
	RMSE(ms)	**69.3**	110.0	146.2
[900, 1200]	MRE(%)	**7.1**	12.1	14.9
	RMSE(ms)	**74.2**	120.7	170.3

Evaluation: Prediction Error

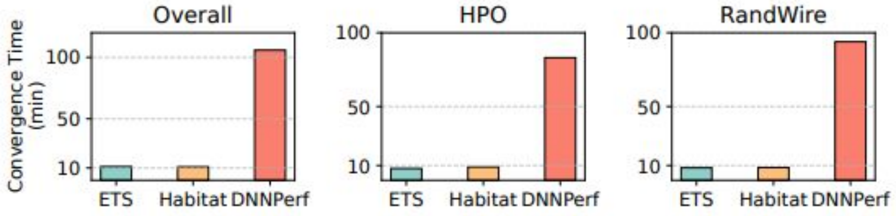


- For most operators; prediction error is low, high only for those with less importance

- The execution trace predicted is very close to the actual trace



Evaluation: Convergence Time

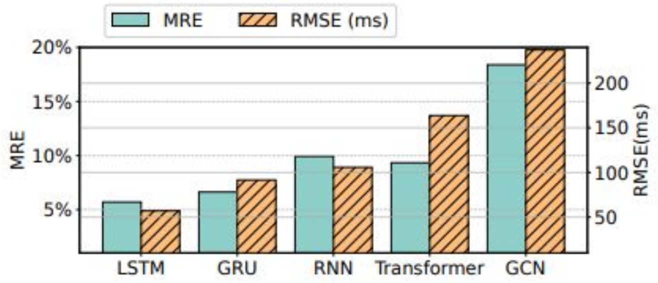


ETS achieves significantly better convergence time compared to *DNNPerf*.

This is because *DNNPerf* is a graph based technique that tries to accommodate diverse model sizes to a single predictor

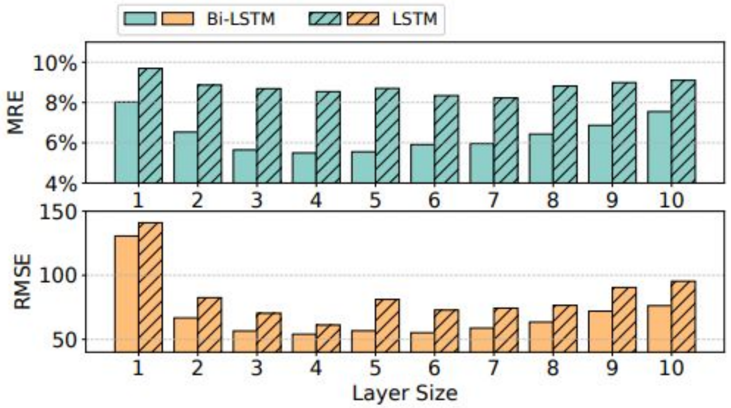
Evaluation: model structure

Comparing **LSTM, GRU, RNN, Transformers and GCN**
LSTM was found to be the best performing model



LSTM tuning showed that:

- Bi-LSTM performs better compared to regular LSTM
- MRE and RMSE reduces up to 4 layers and starts gradually increasing

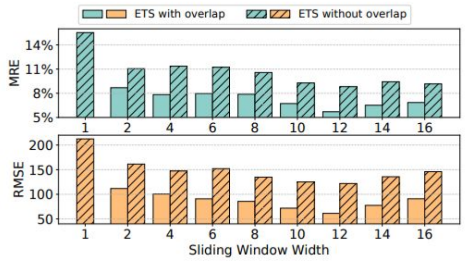


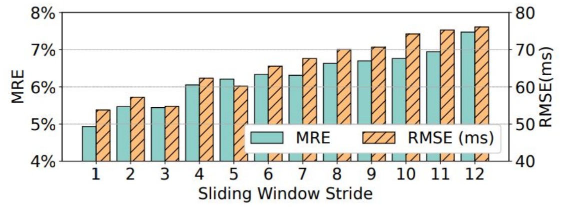
Evaluation: Sliding Window configurations

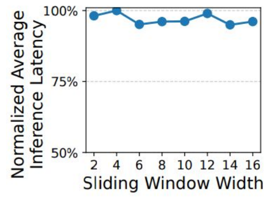
Window overlap (stride half of window width) provides better performance compared to no overlap

MRE and RMSE improve up to sliding window width of 12 with no improvement on further increase

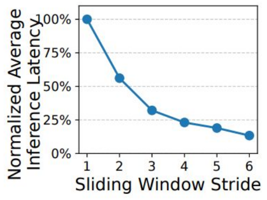
Lower sliding window provides better performance but higher inference latency. Stride of 3 is selected for reasonable accuracy with significantly lower inference latency







(a) Width.



(b) Stride.

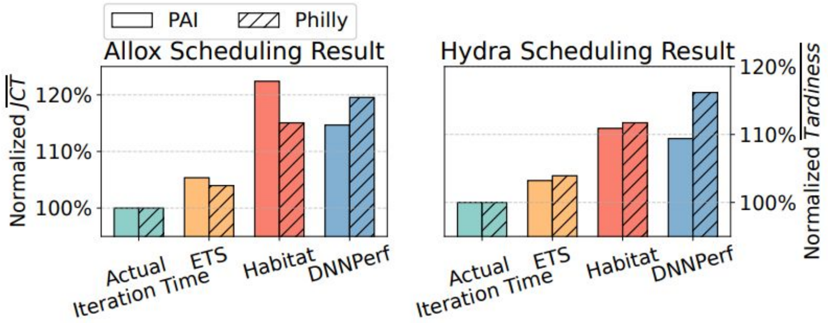
Evaluation: Effect on Scheduling

Job traces: **PAI** and **Philly**

Scheduling: **Allox** and **Hydra**

Metrics:

JCT (Job completion Time) and **Tardiness**



Results:

ETS is most closer to the actual iteration time while other approaches result in higher JCT and Tardiness

Contributions

- Introduce a novel sequential feature extraction technique for DL model training
- Establish a dataset encompassing 14000+ DL model configurations and employ best practice methods for training a prediction model
- Perform extensive evaluations of the proposed approach to build a prediction workflow that surpasses state-of-the-art methods, achieving 5.9% prediction error with only 10 minutes convergence time. It also enhances scheduling outcomes by reducing job completion time by 17%.

Limitations

- Parallelism in deep learning training is very important for large model training which is not considered in this study.

- Comparing different execution environments require the model to be trained for each new environment, while other approaches like Habitat does not need it.

- This study considers the sequential execution scenario where the GPU may not be fully utilized. Resource underutilization results in sub-optimal performance which is generally not desired.

Thank You

Questions