ICML

Nesterov Method for Asynchronous Pipeline Parallel Optimization

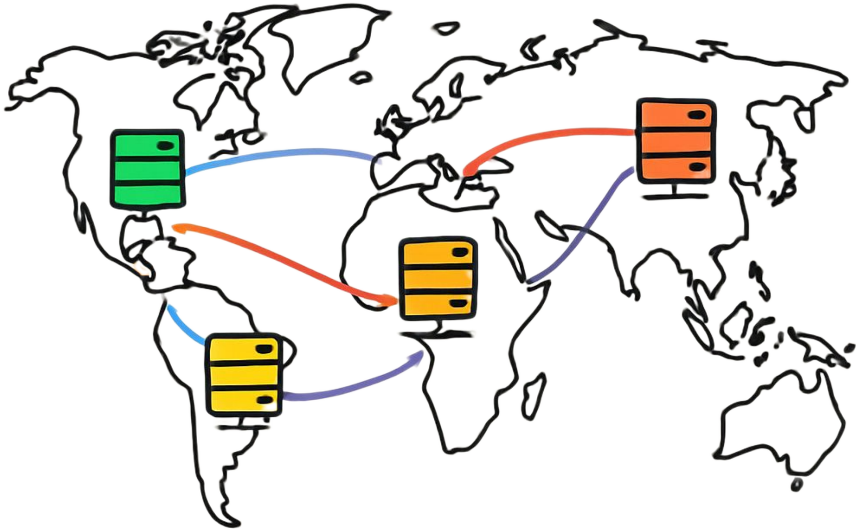
Thalaiyasingam Ajanthan, Sameera Ramasinghe, Yan Zuo, Gil Avraham & Alexander Long

Pluralis Research

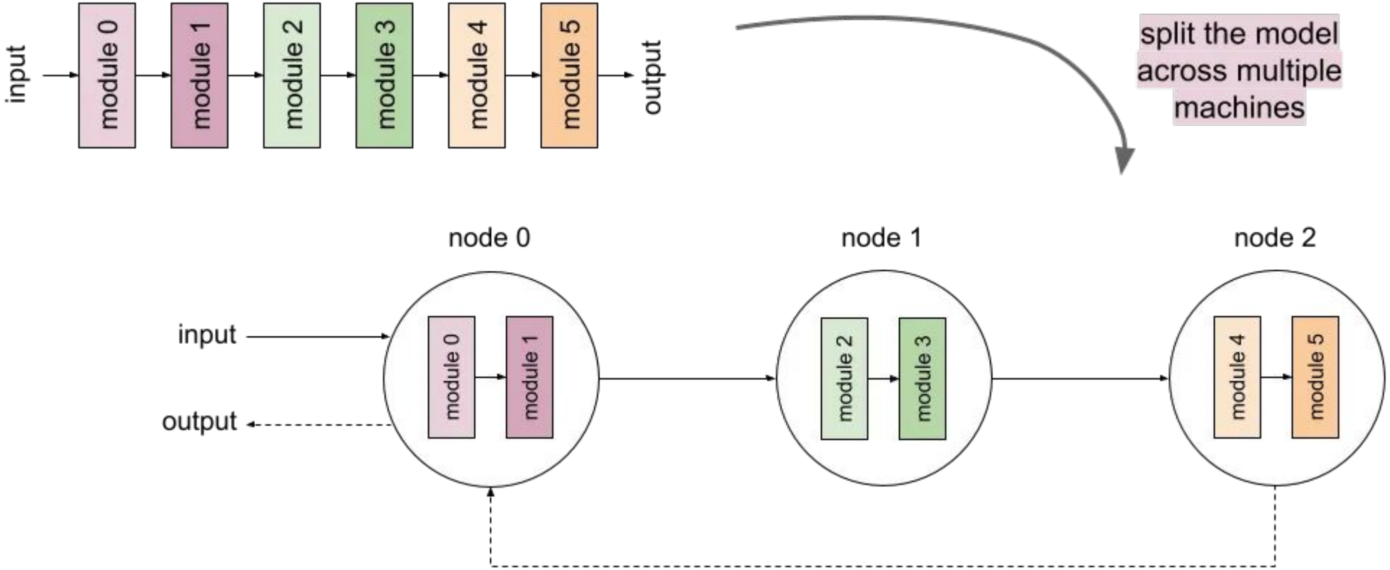
16th July 2025

Asynchronous Training in Distributed Networks

- Distributed training infrastructure is heterogeneous and low-bandwidth
- Synchronous training is prone to "sync bubbles" – idle time
- Asynchronous training eliminates these bubbles
- Though challenging in both DP, PP



Pipeline Parallel



- Pipeline Parallel splits the model across the layer dimension over multiple devices
- This approach was initially designed for synchronous setup

Image credit: https://battox.medium.com/pipeline-parallelism-in-pytorch-dc439f7573e9

Pipeline Parallel: Scheduling

- Pipeline Parallel research focused on synchronous systems
- Methods propose various scheduling to reduce "bubbles"/reduce memory requirements
- Gpipe, 1F1B, Interleaved 1F1B, ZeroBubble, etc.

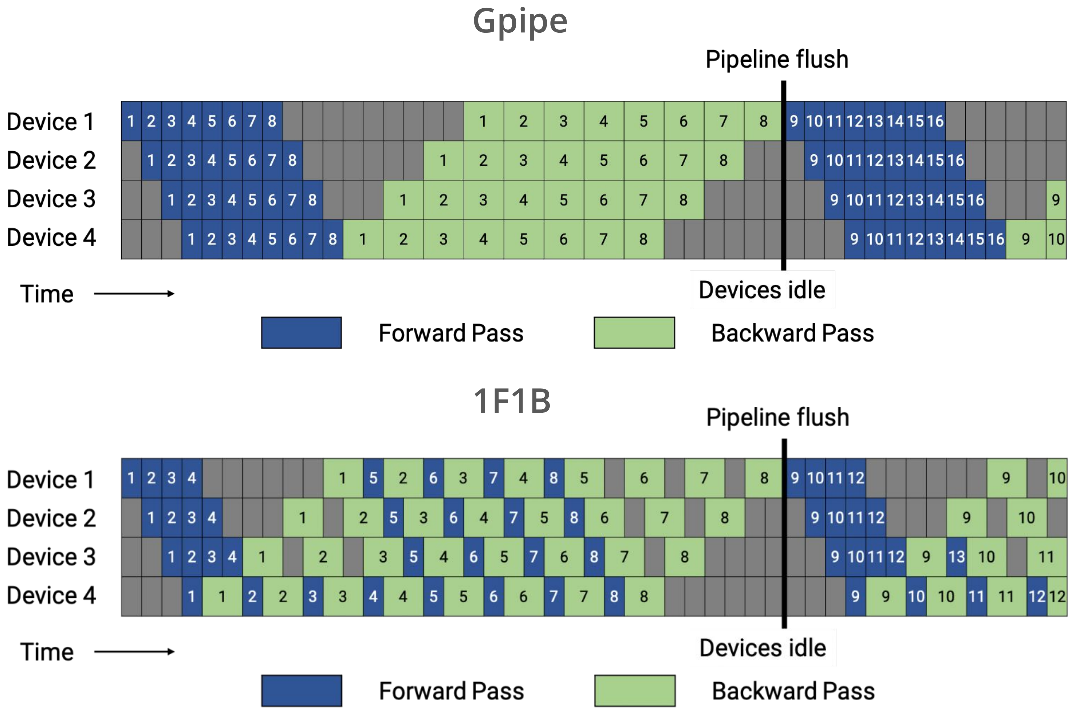


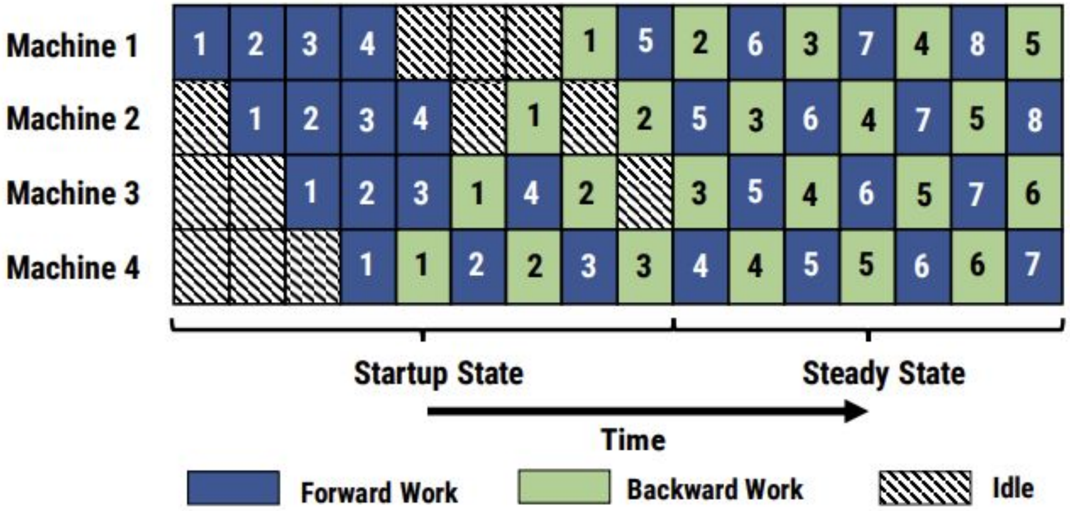
Image credit: https://developer.nvidia.com/blog/scaling-language-model-training-to-a-trillion-parameters-using-megatron/

Pipeline Parallel: Asynchronous

PR

- Asynchronous Pipeline Parallel update stage weights without waiting
- These methods have **no pipeline flush bubble**
- Issues such as stale gradients and outdated weights
- Notable methods: Pipedream, PipeMare

Pipedream



Each machine alternates between forward and backward passes asynchronously

Image credit: https://arxiv.org/pdf/1806.03377

Pipeline Parallel

Formally, for a model split into P pipeline stages

$$F(\mathbf{W}, \mathbf{x}_0) \coloneqq f_P \circ f_{P-1} \circ \cdots \circ f_1(\mathbf{x}_0)$$

Similar, for the backwards:

$$G(\mathbf{W}, \mathbf{e}_P) \coloneqq g_1 \circ g_2 \circ \cdots \circ g_P(\mathbf{e}_P)$$

Where e_P is the error signal at stage P

PR

No discrepancy between weights and gradients

$$\nabla f_i(\mathbf{w}_i^t) = h_i(\mathbf{w}_i^t, \mathbf{e}_i^t)$$
$$\mathbf{e}_{i-1}^t = g_i(\mathbf{w}_i^t, \mathbf{e}_i^t)$$

$\longrightarrow$

$$\mathbf{w}_i^{t+1} = \mathbf{w}_i^t - \eta \nabla f_i(\mathbf{w}_i^t)$$

$$\left\{\begin{aligned} F(\mathbf{W}, \mathbf{x}_0) &:= f_P \circ f_{P-1} \circ \cdots \circ f_1(\mathbf{x}_0) \\ G(\mathbf{W}, \mathbf{e}_P) &:= g_1 \circ g_2 \circ \cdots \circ g_P(\mathbf{e}_P) \end{aligned}\right\}$$

Asynchronous Pipeline Parallel: PipeDream

Gradients are delayed, causing incorrect weight updates

$$\nabla f_i(\mathbf{w}_i^t) = h_i(\mathbf{w}_i^t, \mathbf{e}_i^t) \quad\longrightarrow\quad \nabla f_i({\mathbf{w}_i}^{t-\tau_i}) = h_i({\mathbf{w}_i}^{t-\tau_i}, {\mathbf{e}_i}^{t-\tau_{i+1}})$$

$$\mathbf{e}_{i-1}^t = g_i(\mathbf{w}_i^t, \mathbf{e}_i^t) \quad\longrightarrow\quad {\mathbf{e}_{i-1}}^{t-\tau_i} = g_i({\mathbf{w}_i}^{t-\tau_i}, {\mathbf{e}_i}^{t-\tau_{i+1}})$$

$$\mathbf{w}_i^{t+1} = \mathbf{w}_i^t - \eta \nabla f_i(\mathbf{w}_i^t) \quad\longrightarrow\quad \mathbf{w}_i^{t+1} = \mathbf{w}_i^t - \eta \nabla f_i({\mathbf{w}_i}^{t-\tau_i})$$

Narayanan et al, Pipedream: Generalized pipeline parallelism for dnn training

Nesterov Accelerated Gradient (NAG)

PR

- Nesterov Accelerated Gradient (NAG) has optimal $O(\frac{1}{t^2})$ convergence rate for smooth convex functions in non stochastic settings
- The momentum term γ_t satisfies $\gamma_1 = 0, 0 < \gamma_t < 1$
- NAG intuition: reduces overshooting and stabilizes training by computing the gradients in velocity adjusted step (look-ahead)

$$\mathbf{d}_t = \gamma_t(\mathbf{w}_t - \mathbf{w}_{t-1})$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{d}_t - \eta \nabla f(\mathbf{w}_t + \mathbf{d}_t)$$

Yurii Nesterov. Introductory lectures on convex optimization: A basic course

NAG for Delay Correction

- Recall for async Pipeline Parallel:

$$\nabla f_i(\mathbf{w}_i^t) = h_i(\mathbf{w}_i^t, \mathbf{e}_i^t) \qquad \longrightarrow \qquad \nabla f_i({\mathbf{w}_i}^{t-\tau_i}) = h_i({\mathbf{w}_i}^{t-\tau_i}, {\mathbf{e}_i}^{t-\tau_{i+1}})$$

$$\mathbf{e}_{i-1}^t = g_i(\mathbf{w}_i^t, \mathbf{e}_i^t) \qquad \qquad {\mathbf{e}_{i-1}}^{t-\tau_i} = g_i({\mathbf{w}_i}^{t-\tau_i}, {\mathbf{e}_i}^{t-\tau_{i+1}})$$

- Incorrect weight update stems from **stale gradients**

- If the weight delay is defined as:

$$\bar{\mathbf{w}}_t = \mathbf{w}_{t-\tau} = \mathbf{w}_t - \Delta_t \ , \qquad \bar{\mathbf{d}}_t = \mathbf{d}_{t-\tau} \ .$$

- <u>Intuition</u>: NAG look-ahead operates as delay correction:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{d}_t - \eta(1 - \gamma_t)\nabla f(\bar{\mathbf{w}}_t + \bar{\mathbf{d}}_t) \ .$$

NAG for Delay Correction

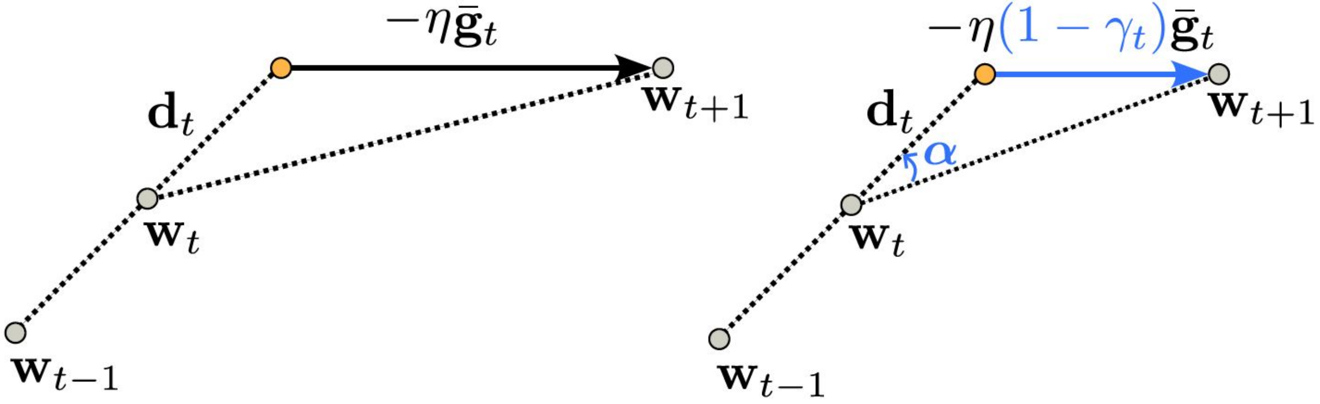
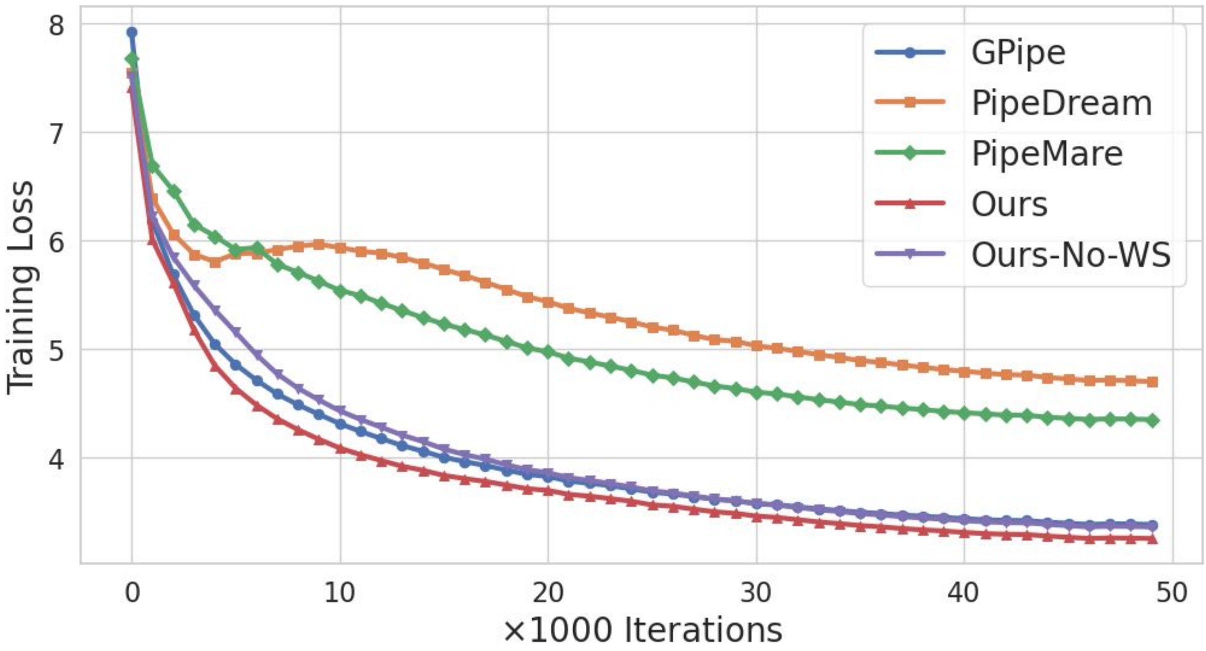


Figure 1: *Original NAG (left) and our modified version (right) for delayed gradients (denoted with $\bar{\mathbf{g}}_t$). Our method discounts the gradient term by $(1-\gamma_t)$. When $\gamma_t \to 1$, the angle $\alpha \to 0$, making the weight trajectory smoother. Consequently, the look-ahead $\mathbf{d}_t$ can be shown to act as delay correction, alleviating gradient staleness.*

Results: NanoGPT on WikiText



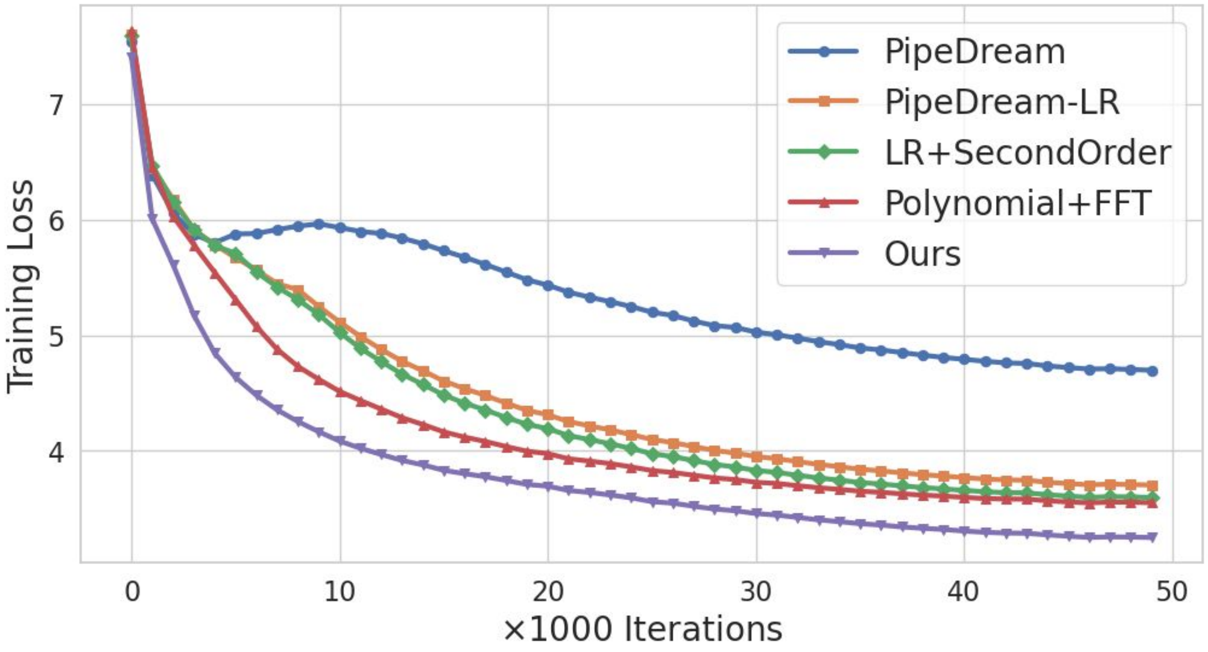
- Our asynchronous method outperforms synchronous GPipe

Results: Multiple Datasets

Method	WT	BC	OWT	Memory
GPipe	30.63	42.39	65.17	$O(N)$
PipeDream	99.48	52.98	224.30	$O(PN)$
PipeMare	71.38	76.93	239.13	$O(N)$
Ours	**27.72**	**39.85**	**62.86**	$O(PN)$
Ours-No-WS	29.90	42.61	108.20	$O(N)$

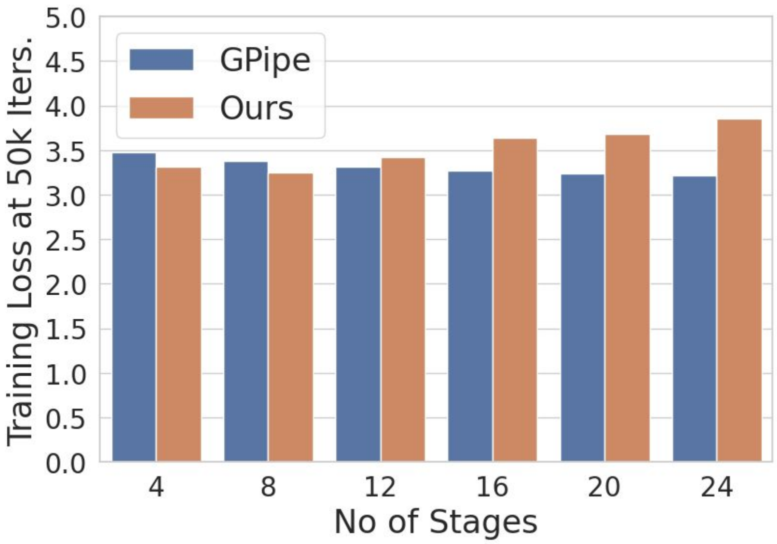
- Our asynchronous method outperforms synchronous GPipe

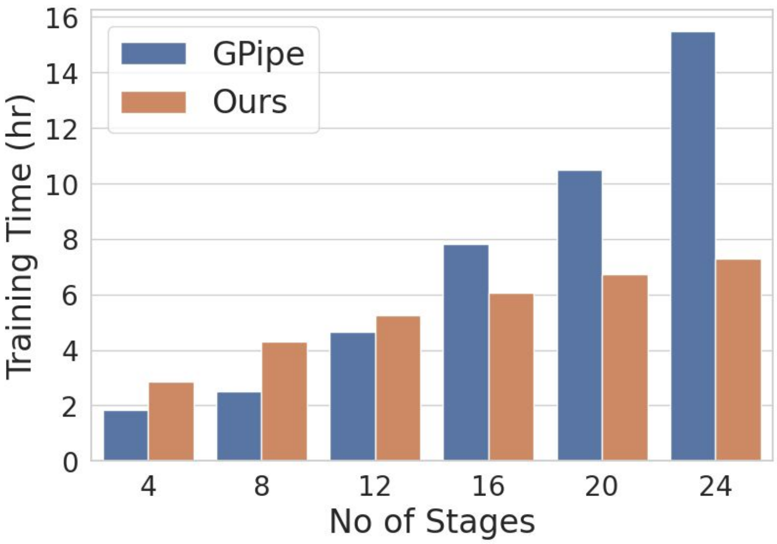
Results: Delay Correction Methods



- Our NAG method outperforms other delay correction methods

Results: Increasing the No of Stages





- Even though, performance slightly degrades for our method compared to GPipe, the training time increase is exponentially larger for GPipe

Thank you