DynaPipe: Optimizing Multi-task Training through Dynamic Pipelines

Chenyu Jiang[1*], Zhen Jia[2], Shuai Zheng[2], Yida Wang[2], Chuan Wu[1]

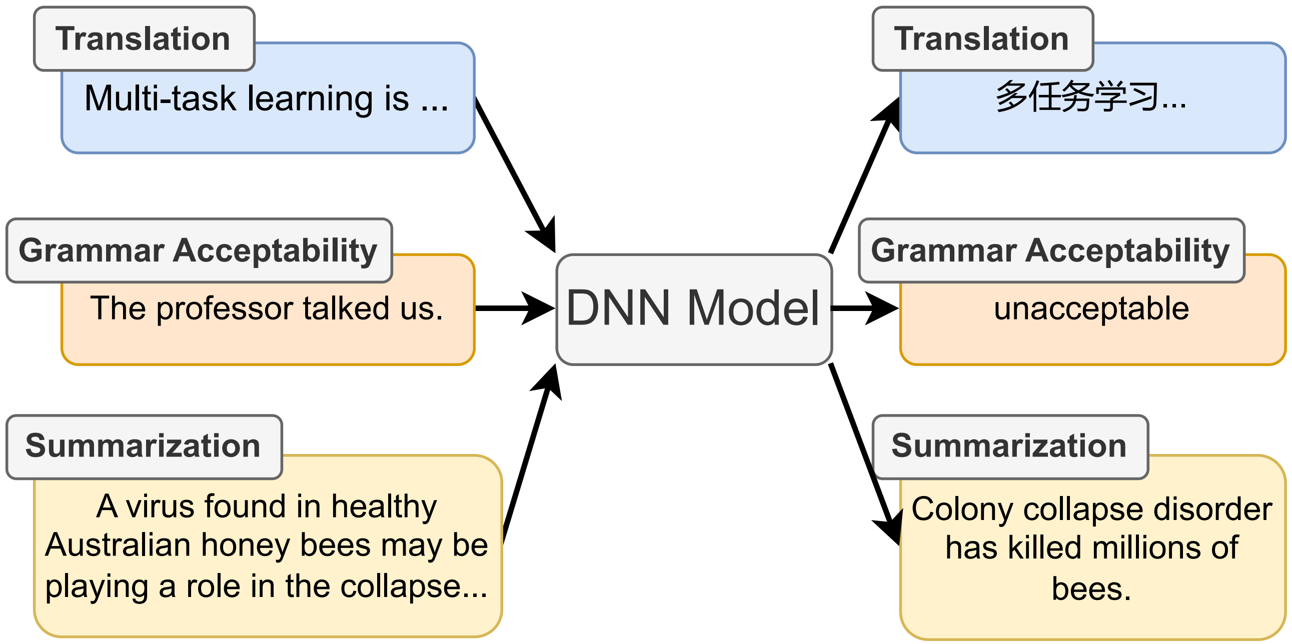
[1]The University of Hong Kong, [2]Amazon Web Services

Accepted to EuroSys 2024

*Work done while interning at AWS.

Background

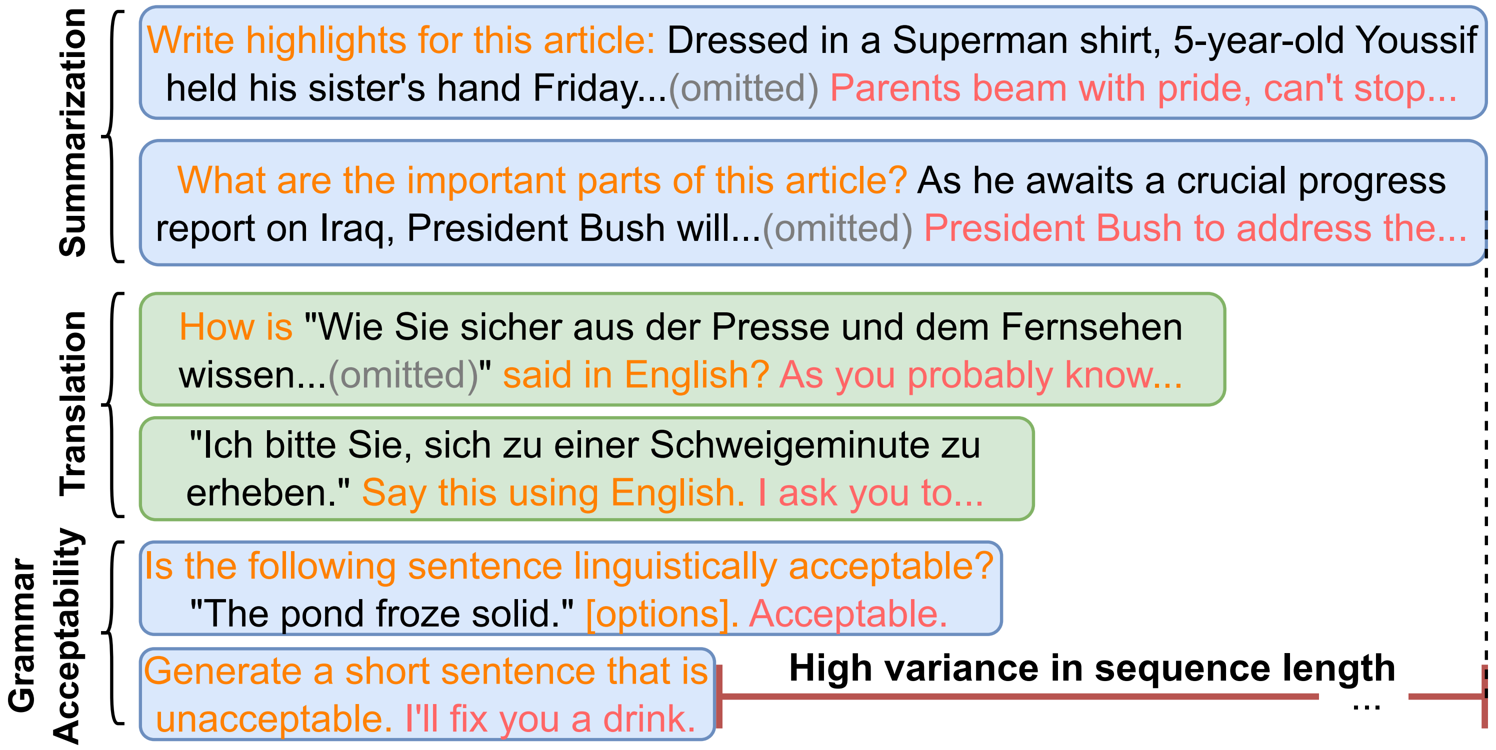
Multi-task Training



One DNN model, multiple tasks.

Background

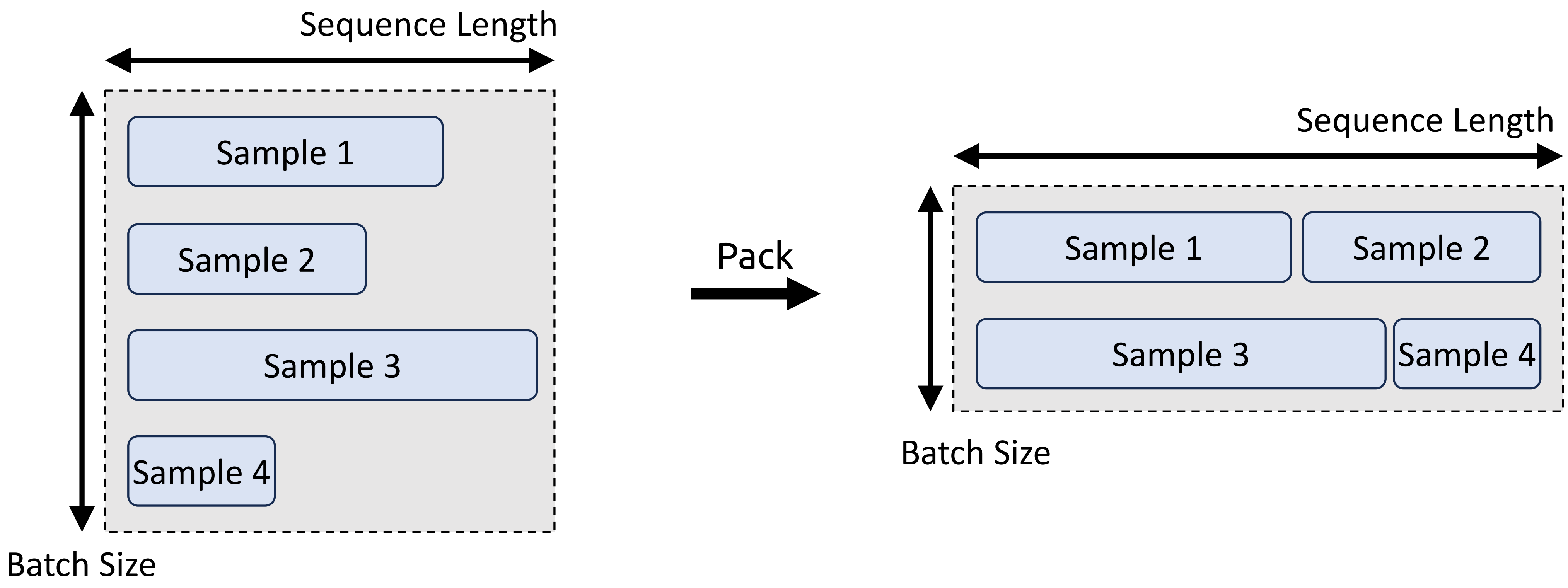
High sequence length variation in multi-task training



Large amount of padding needed.

Background

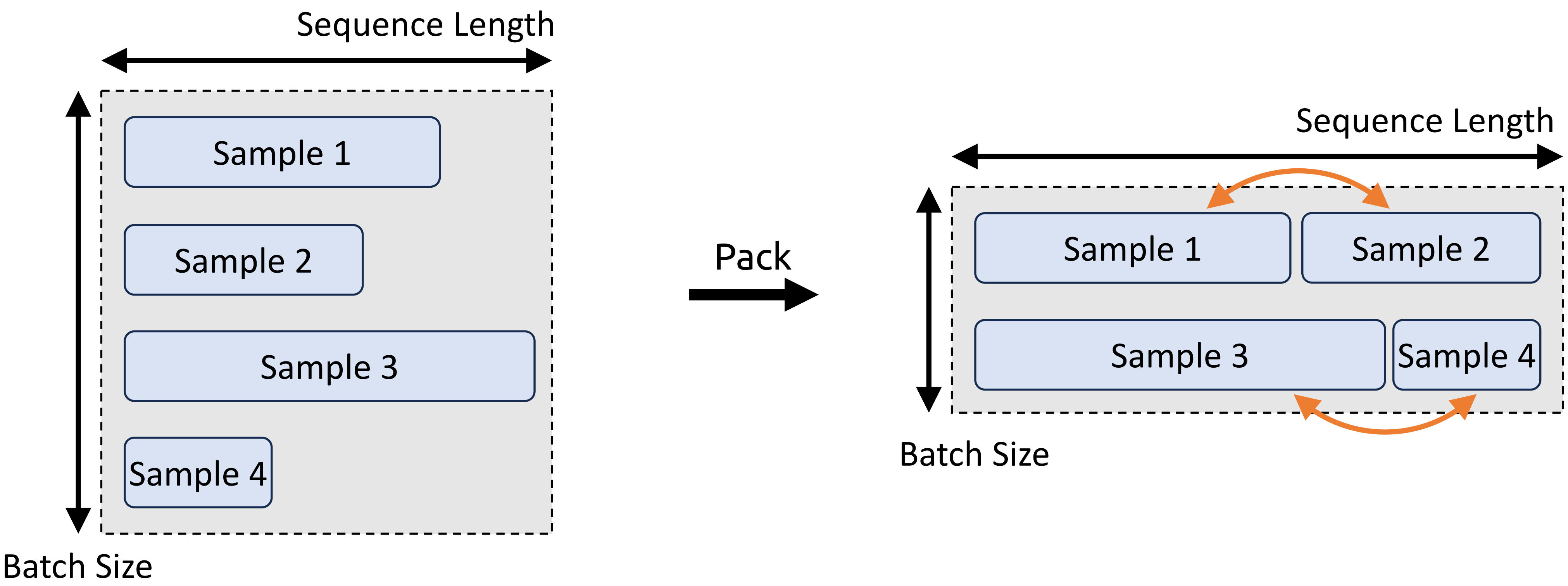
Current solution: packing



Concatenating short sequences to long ones up to fixed maximum length.

Background

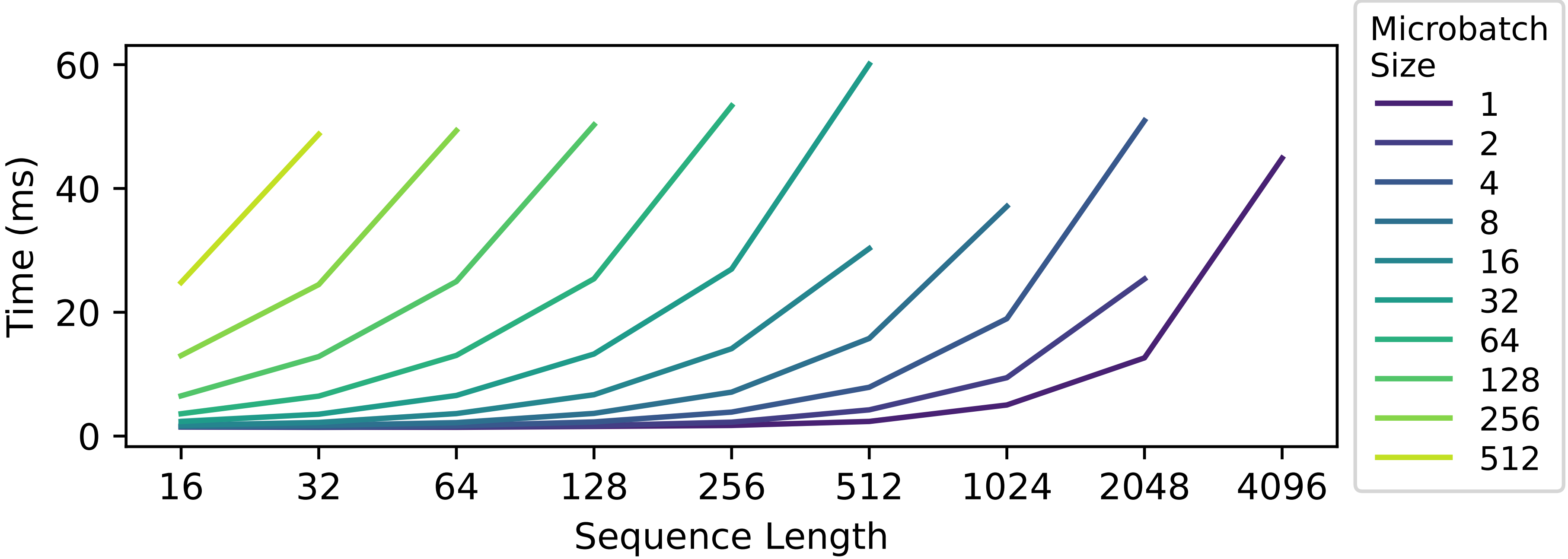
Current solution: packing



Drawback: **unnecessary attention** between unrelated samples.

Background

Current solution: packing



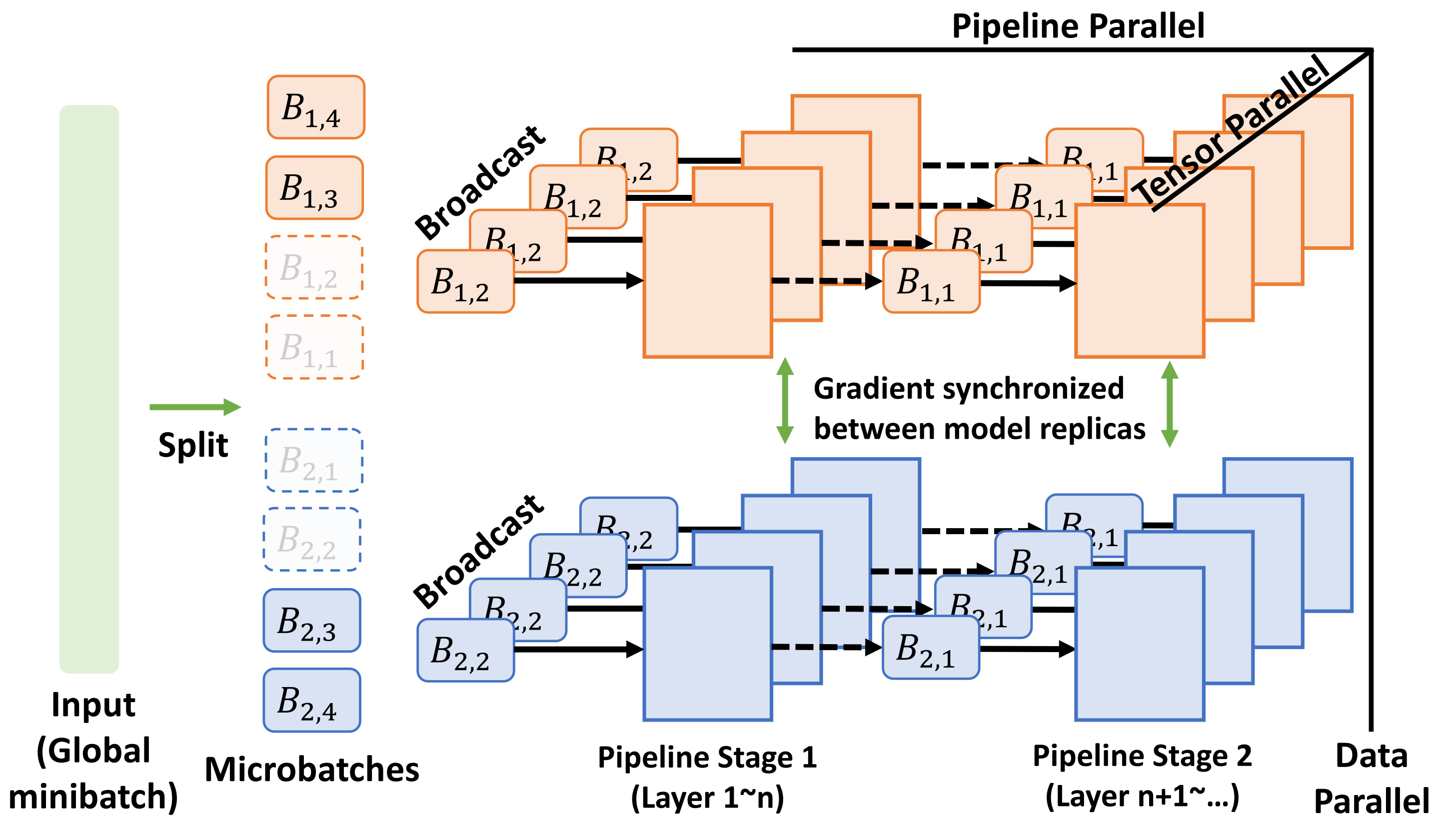
Computation time of a single Transformer encoder layer in T5-11B on an A100 GPU

Drawback: **super-linear** execution time growth with sequence length

Motivation

Proposed solution: dynamic micro-batching

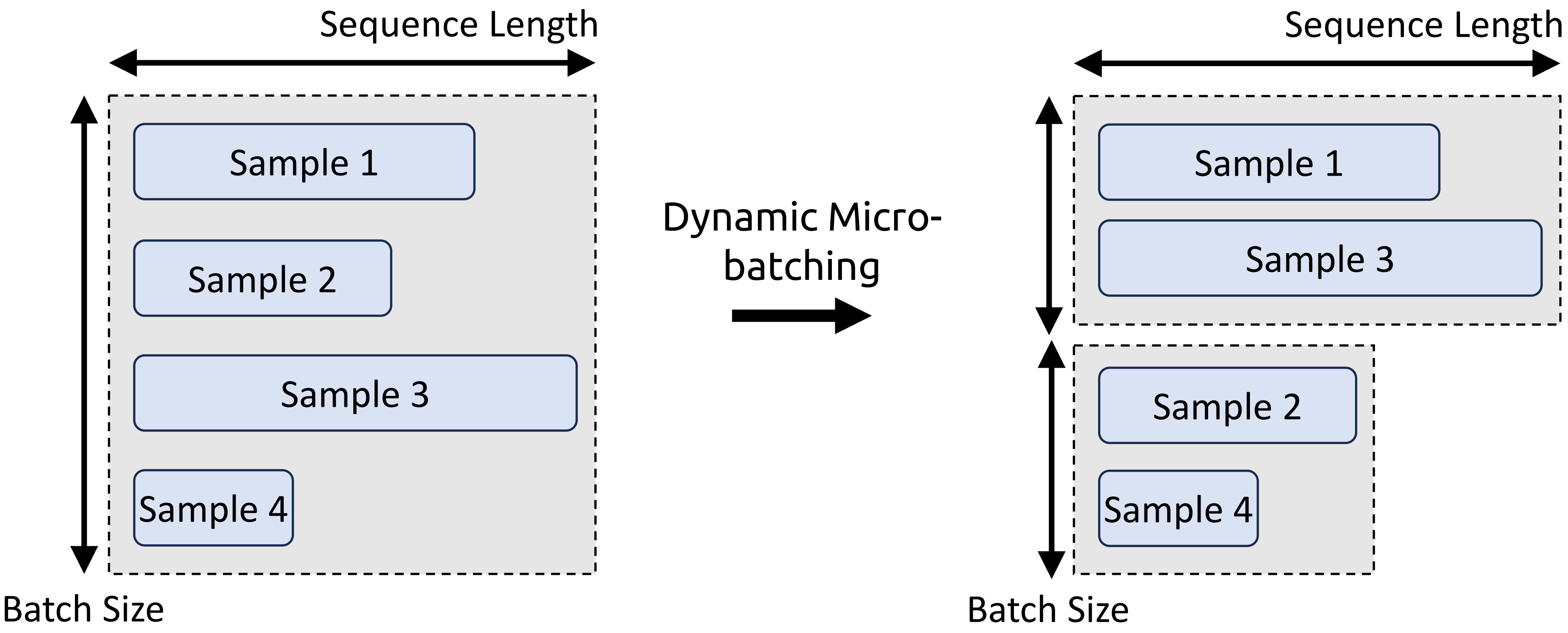
Optimally split each input global batch into micro-batches with similar sequence lengths.



Motivation

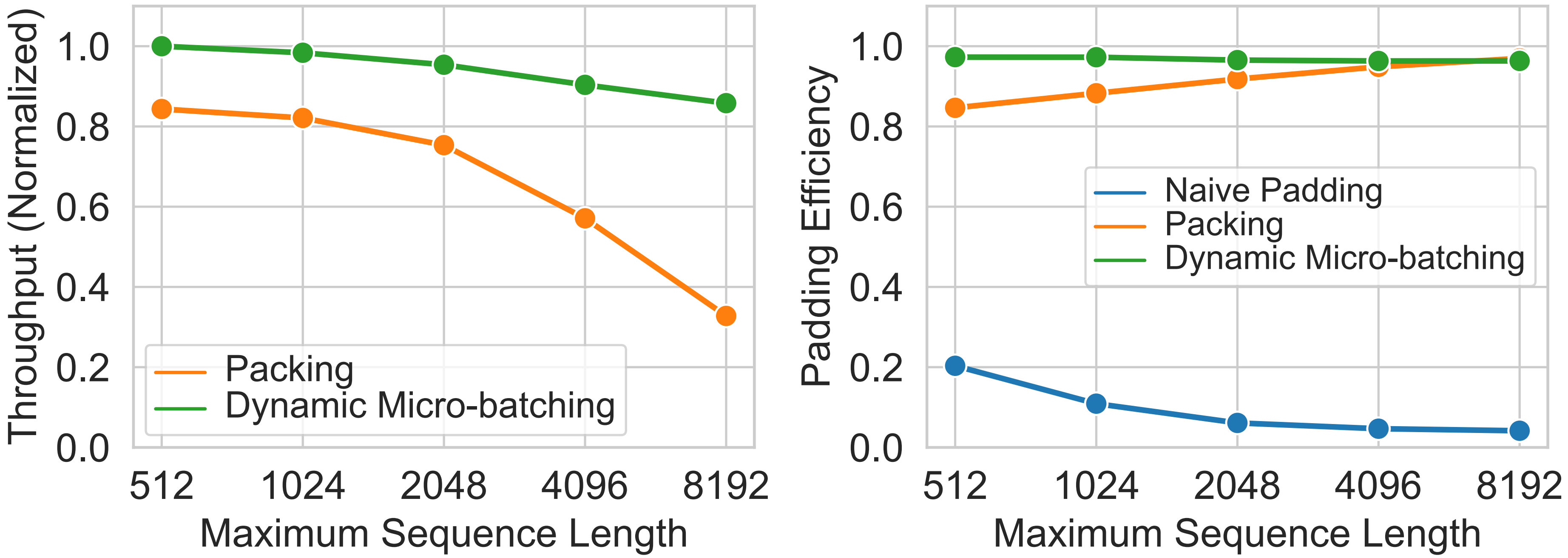
Proposed solution: dynamic micro-batching

Optimally split each input global batch into micro-batches with similar sequence lengths.



Motivation

Proposed solution: dynamic micro-batching

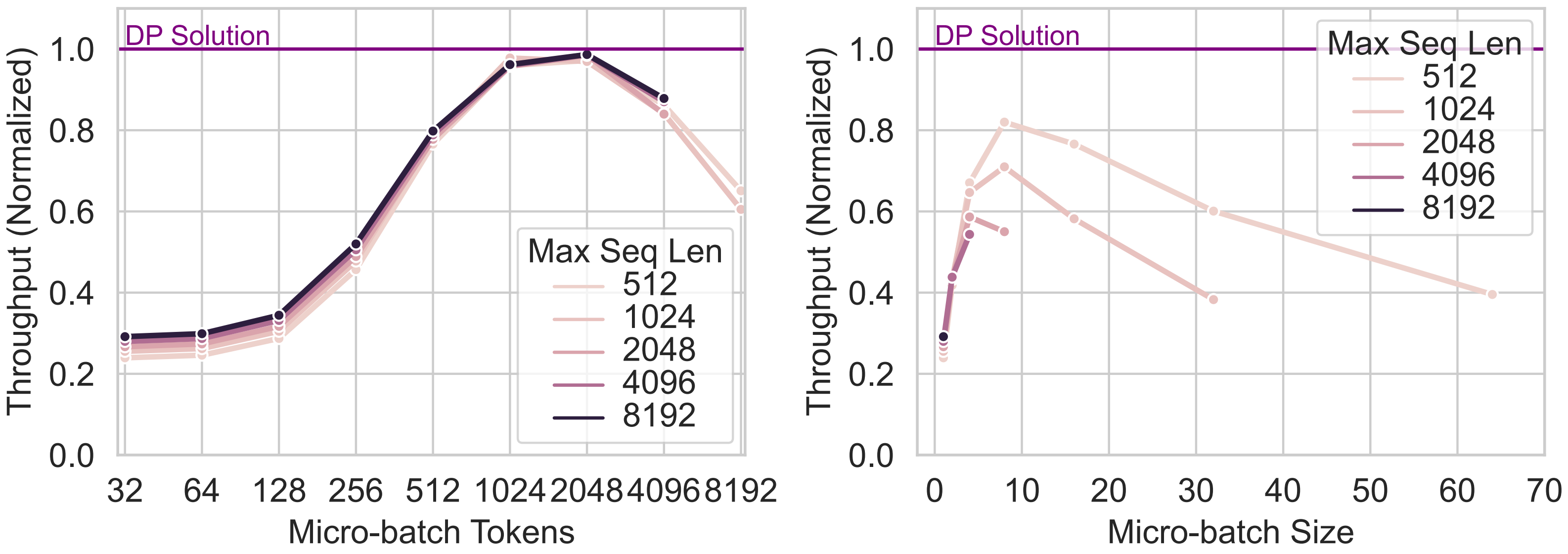


Comparison of throughput and padding efficiency of different methods when training a GPT model with FLANv2 dataset

Motivation

Challenges of dynamic micro-batching

No principled way to split training mini-batches into micro-batches of different sequence lengths.



Training performance of GPT under different micro-batching methods

Micro-batching is critical for performance, yet difficult to find the best batching configuration.

Our solution: a dynamic programming algorithm to optimally construct micro-batches

Motivation

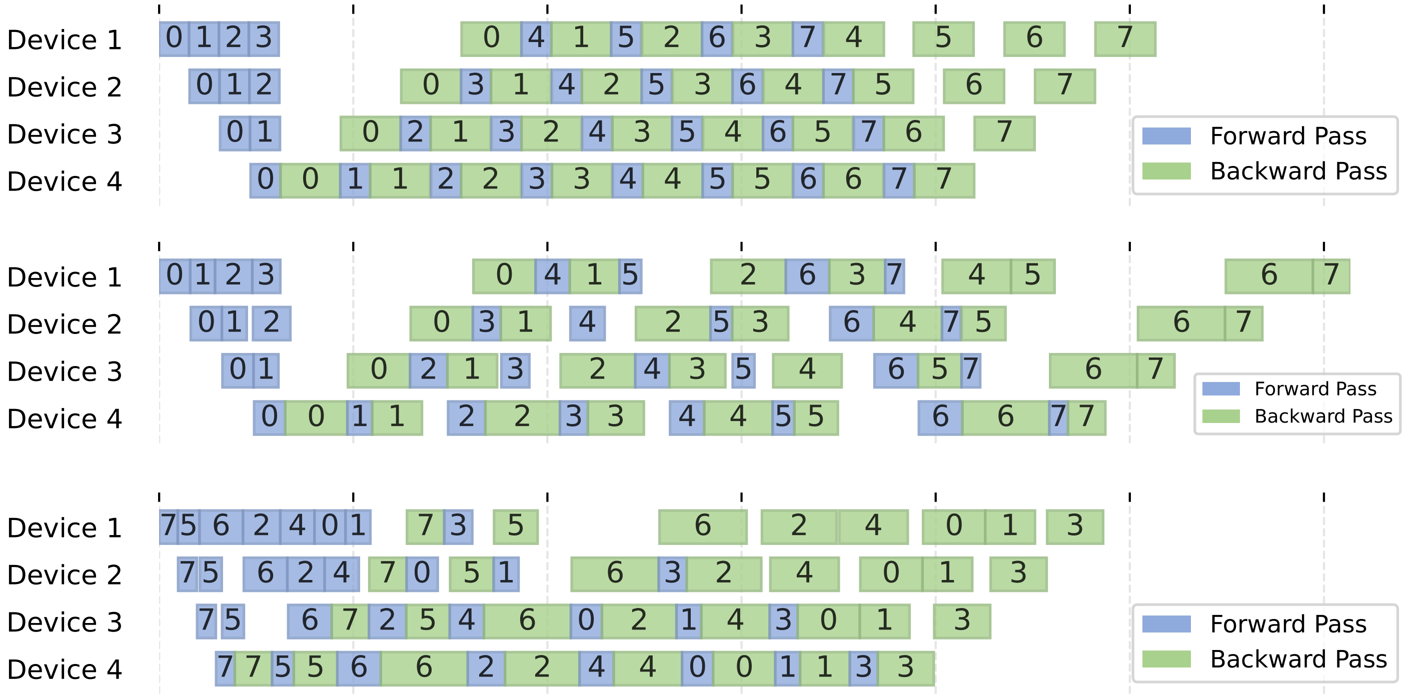
Challenges of dynamic micro-batching

No efficient pipeline schedules for micro-batches of diverse execution times.

1F1B (uniform micro-batches)

1F1B (dynamic micro-batches)

DynaPipe's schedule



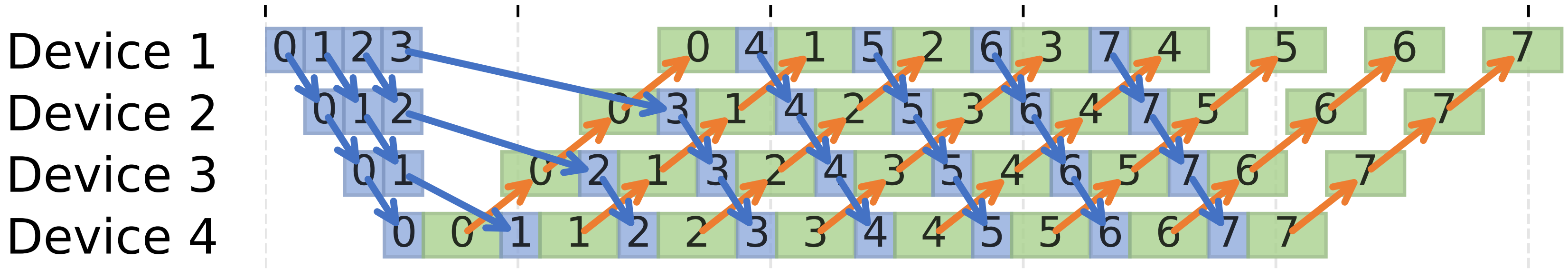
1F1B scheduling is inefficient under dynamic micro-batches.

Our solution: an efficient schedule minimizing GPU idling for pipeline parallel training

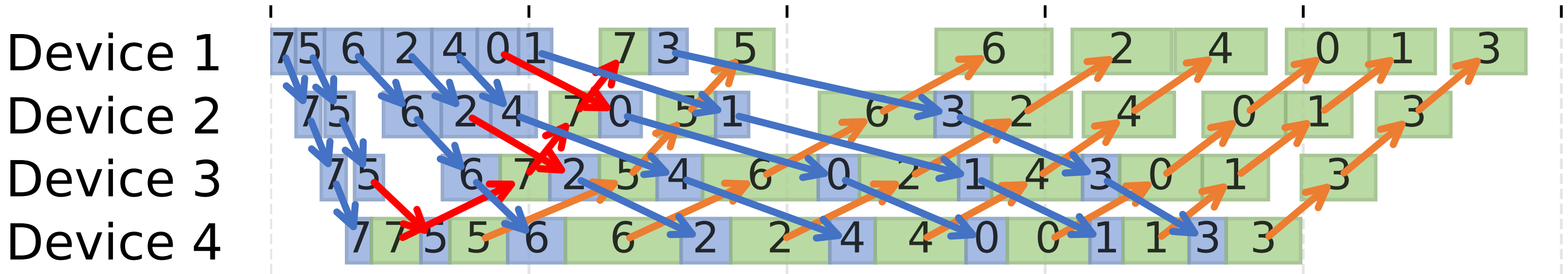
Motivation

Challenges of dynamic micro-batching

Improper communication order between pipeline stages may lead to deadlocks in dynamic pipelines.



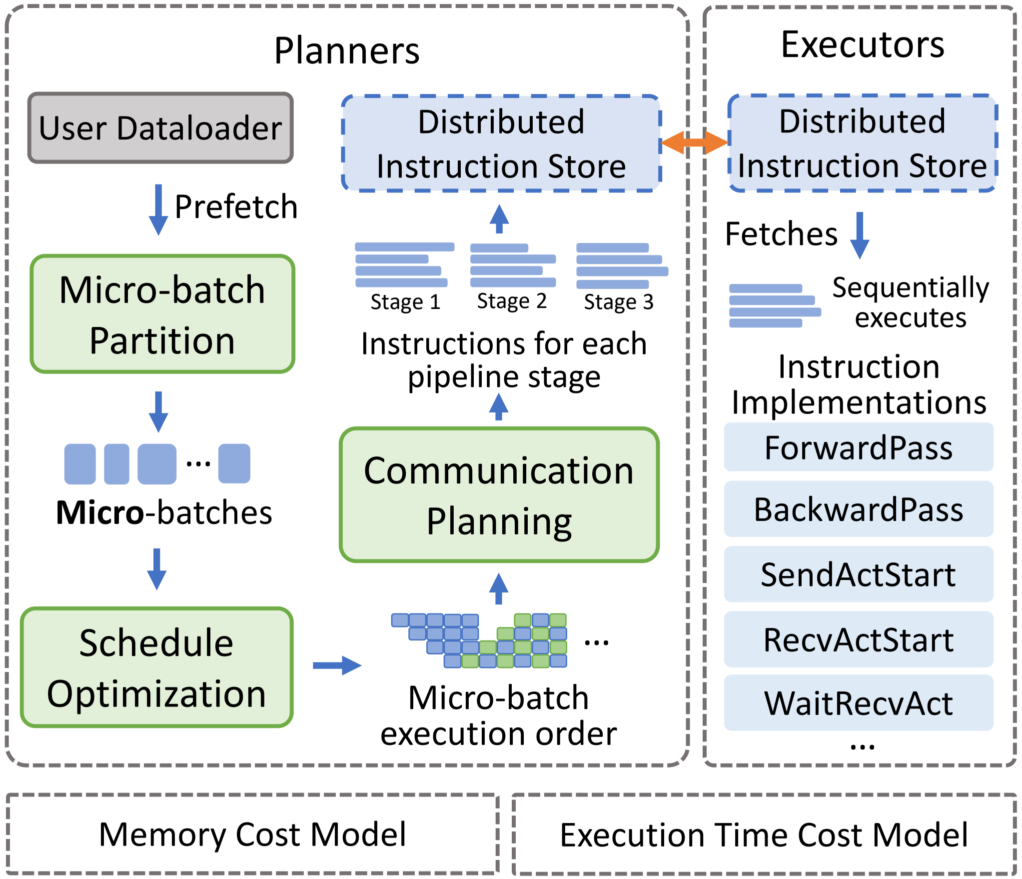
Regular communication pattern in 1F1B schedule



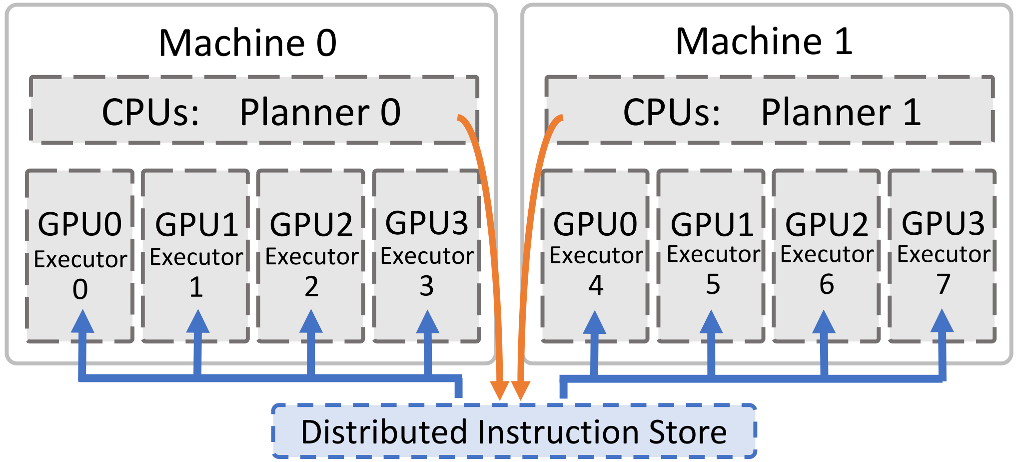
Irregular communication pattern in DynaPipe which changes each iteration

Our solution: plan the communication order for each iteration in advance

Overview



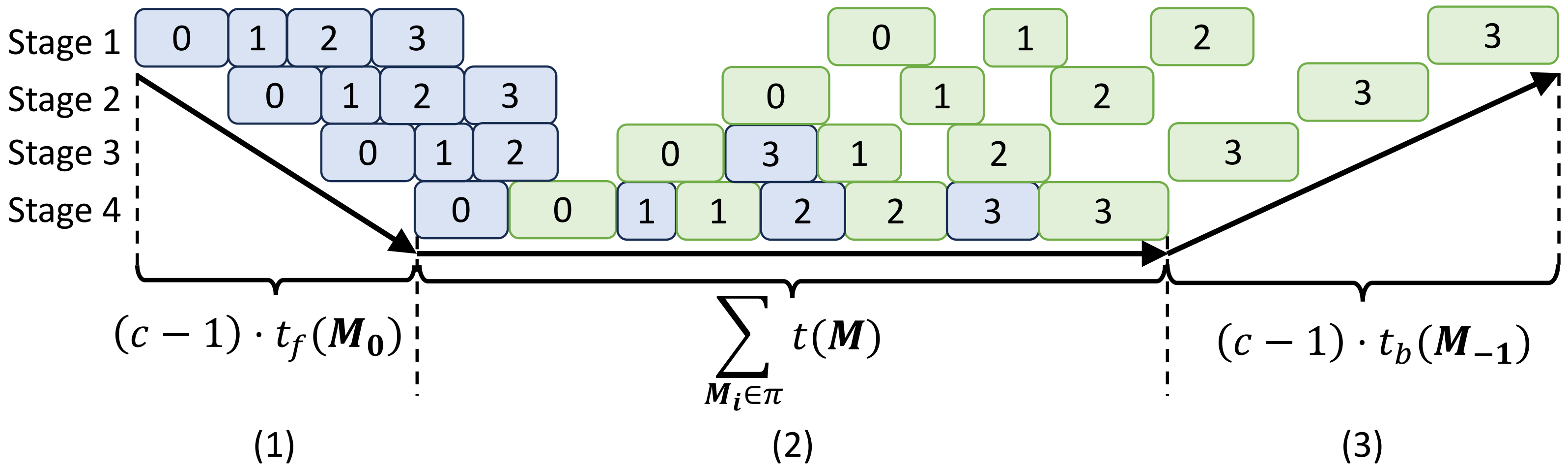
Modules



Physical View

Micro-batch Construction

Modelling pipeline execution time



$$\min_{\boldsymbol{\pi}} \left\{(c-1)\cdot \max\{t(\mathbf{M_i})|\mathbf{M_i}\in\boldsymbol{\pi}\} + \sum_{\mathbf{M_i}\in\boldsymbol{\pi}} t(\mathbf{M_i})\right\}$$

π: The set of micro-batches

c: Number of pipeline stages

$t(M_i)$: execution time of micro-batch M_i

Micro-batch Construction

DP algorithm

$$f(n; t_{max}) = \min_{1 \le i \le n-1} \{f(i; t_{max}) + t(\mathbf{M}_{\mathbf{S[i+1:n]}}) | t(\mathbf{M}_{\mathbf{S[i+1:n]}}) \le t_{max}\}$$

$f(n; t_{max})$: Execution time when optimally partitioning samples 1…n into micro-batches, while each micro-batch's execution time is less than t_{max}.

$t(M_{s[i:j]})$: execution time of micro-batch consisting of samples i…j

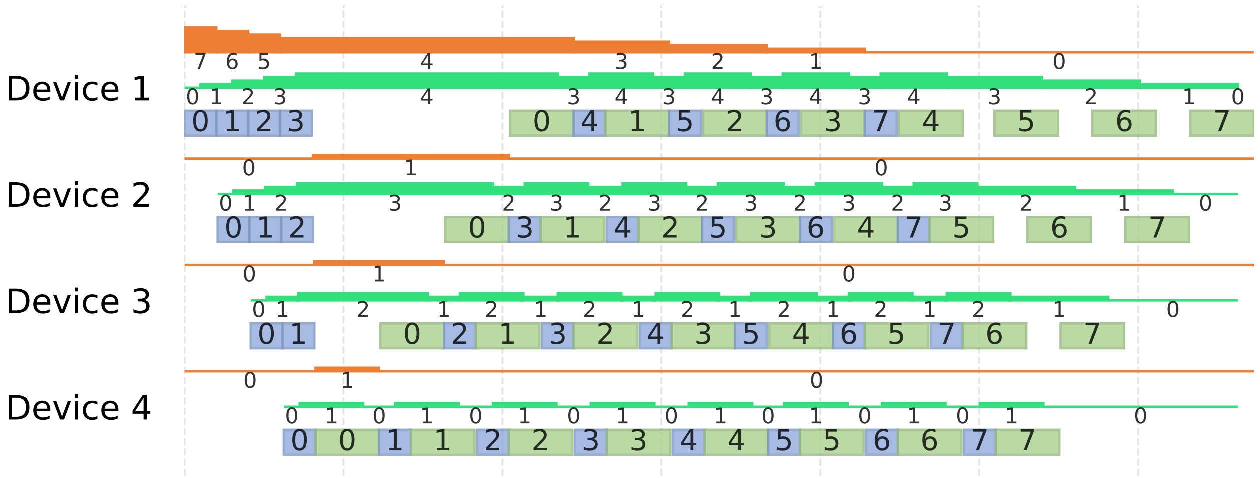
Goal: Find $\min_{t_{max}} f(N; t_{max})$

Pipeline Execution Schedule

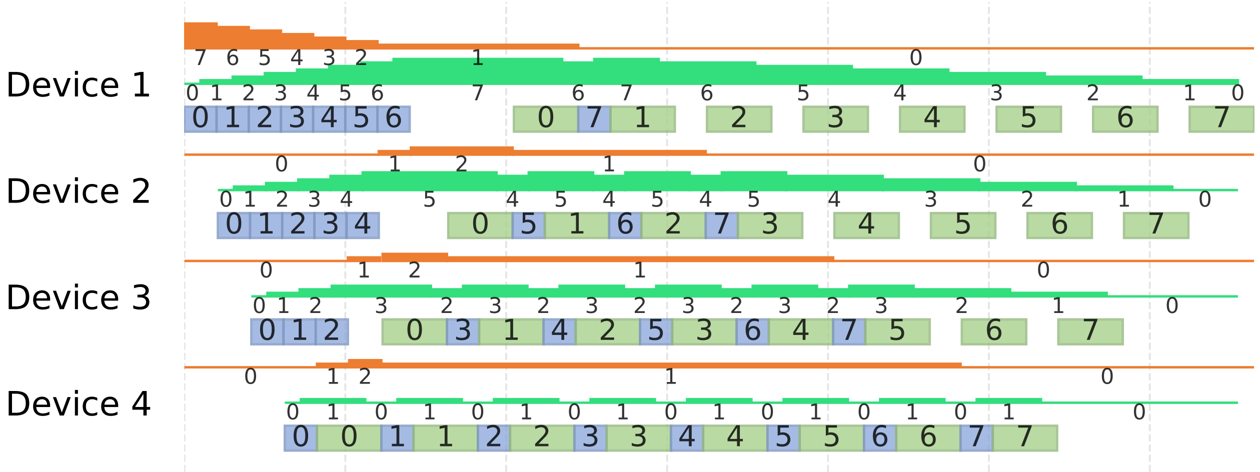
Adaptive schedule: controlling the injection time of micro-batches

Safety Stocks: number of ready (all previous stages have been completed) micro-batches at each device

1F1B: 0 safety stocks during steady state, prone to execution time variation



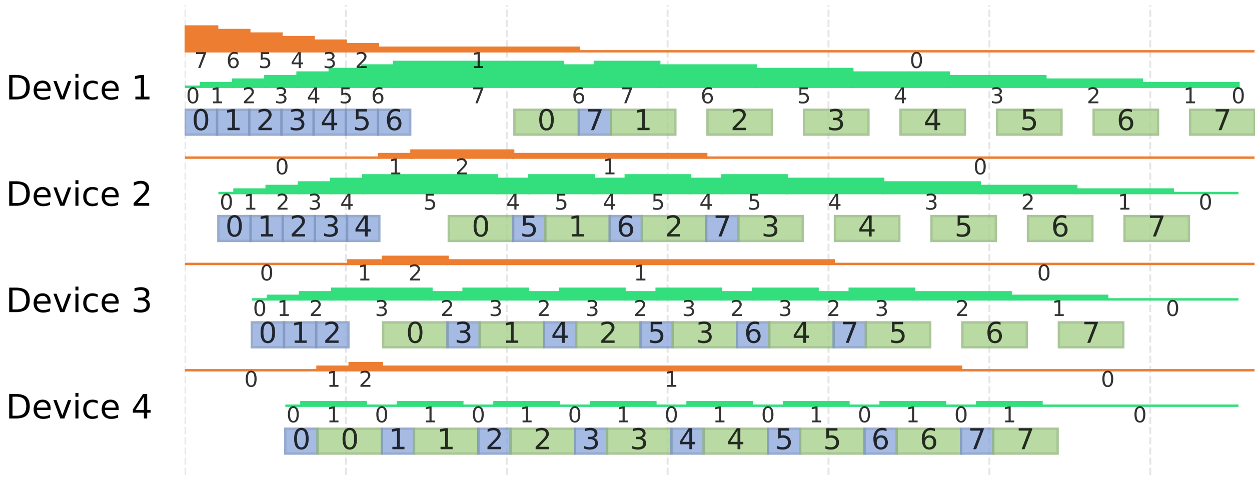
Inject more micro-batches in the beginning of iteration: 1 safety stocks during steady state, more robust to variation



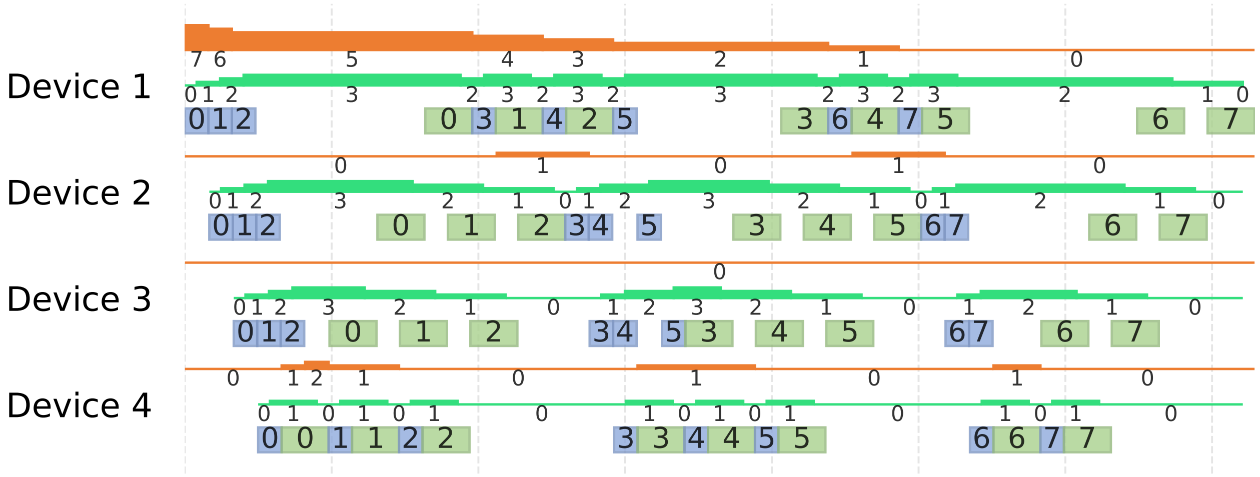
Pipeline Execution Schedule

Adaptive schedule: controlling the injection time of micro-batches

Inject more micro-batches in the beginning of iteration: also consumes more memory

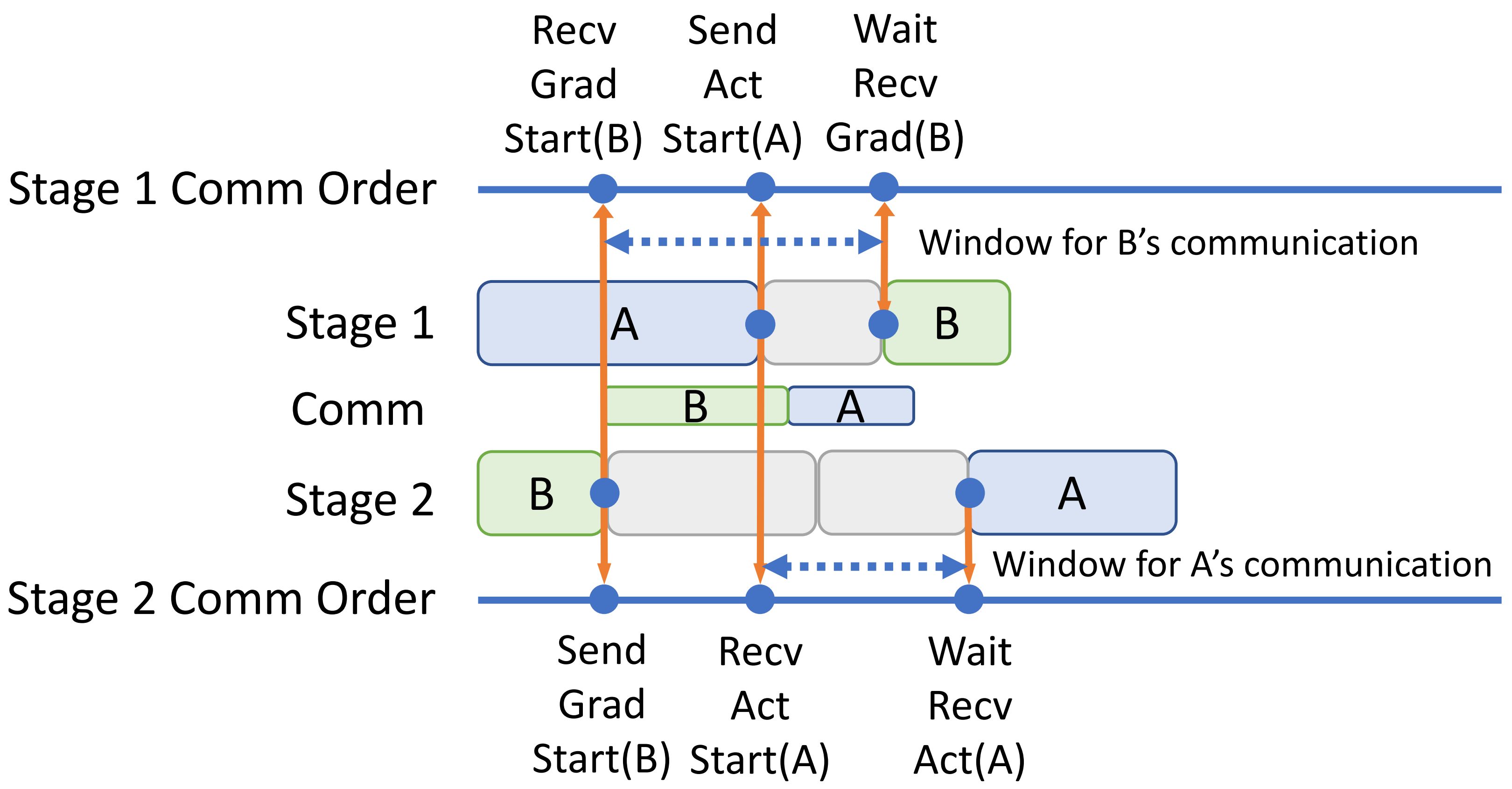


When not enough memory: delay injection of micro-batches, reducing peak memory from 7 micro-batches' activation to 3



Communication Planning

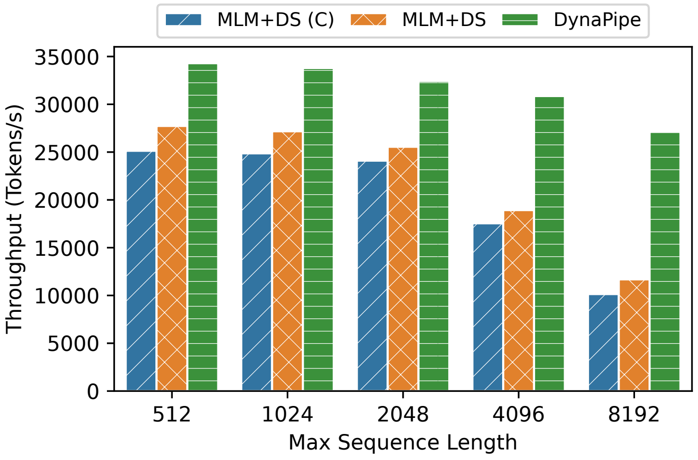
Plan ahead: generate send-recv instruction pairs using simulated timeline



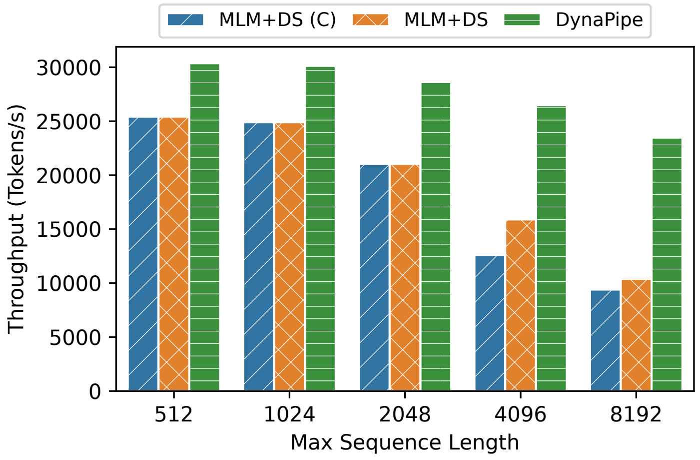
Evaluation

GPT on FLANv2 Dataset

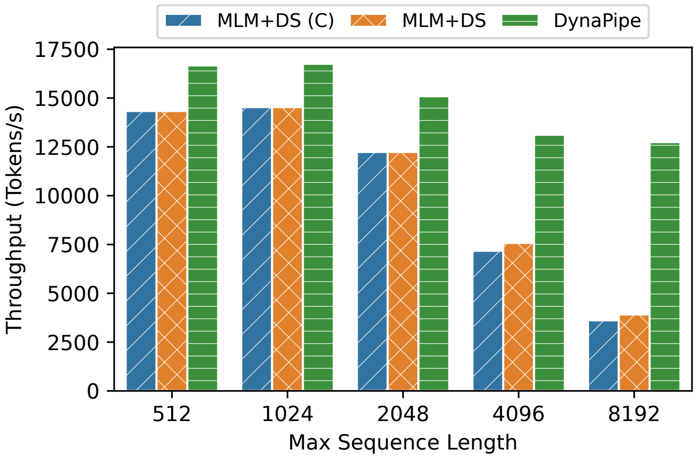
Up to **3.25x** speed up compared to Megatron-LM + DeepSpeed.



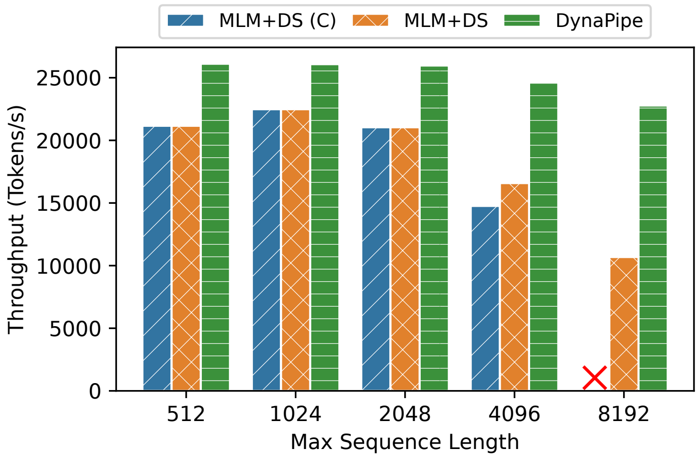
4 A100s



8 A100s



16 A100s

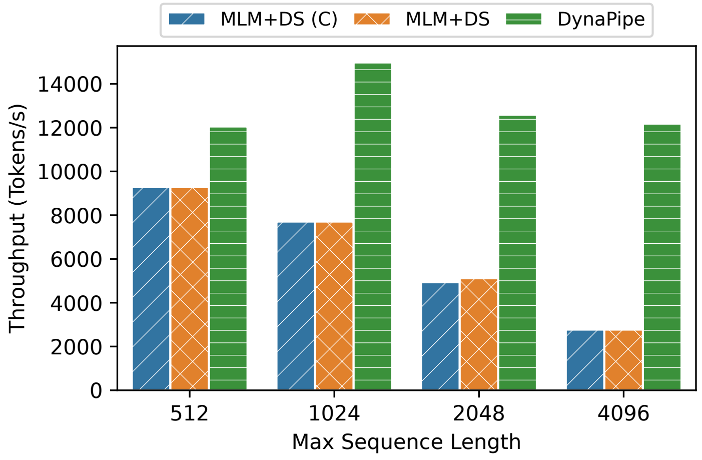


32 A100s

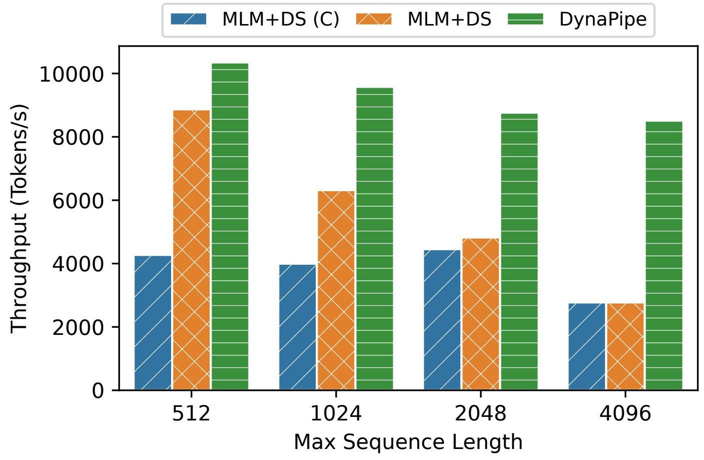
Evaluation

T5 on FLANv2 Dataset

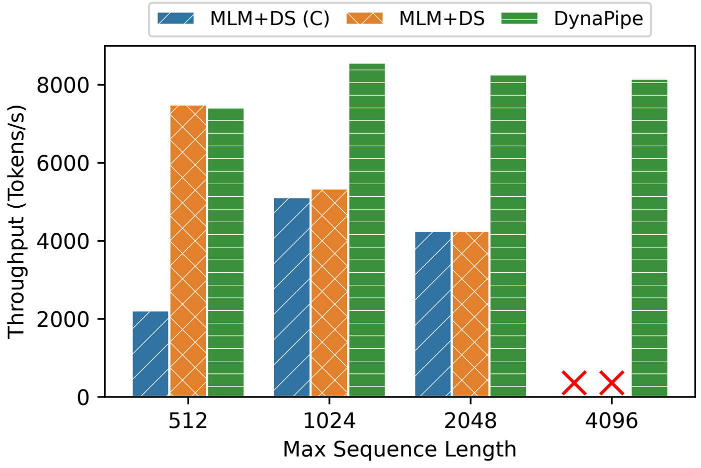
Up to **4.39x** speed up compared to Megatron-LM + DeepSpeed.



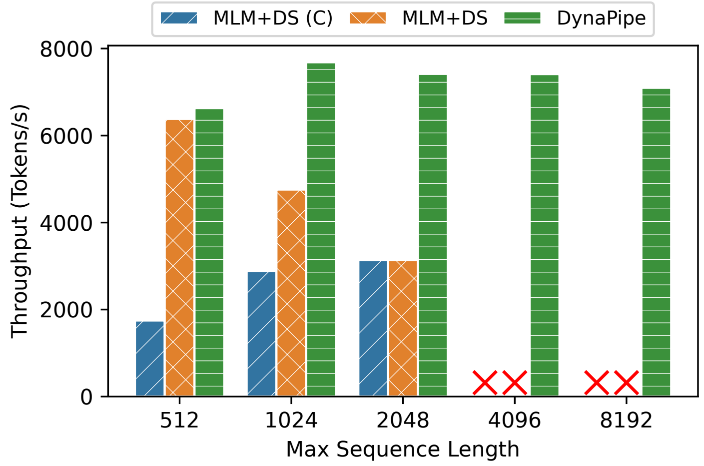
4 A100s



8 A100s



16 A100s

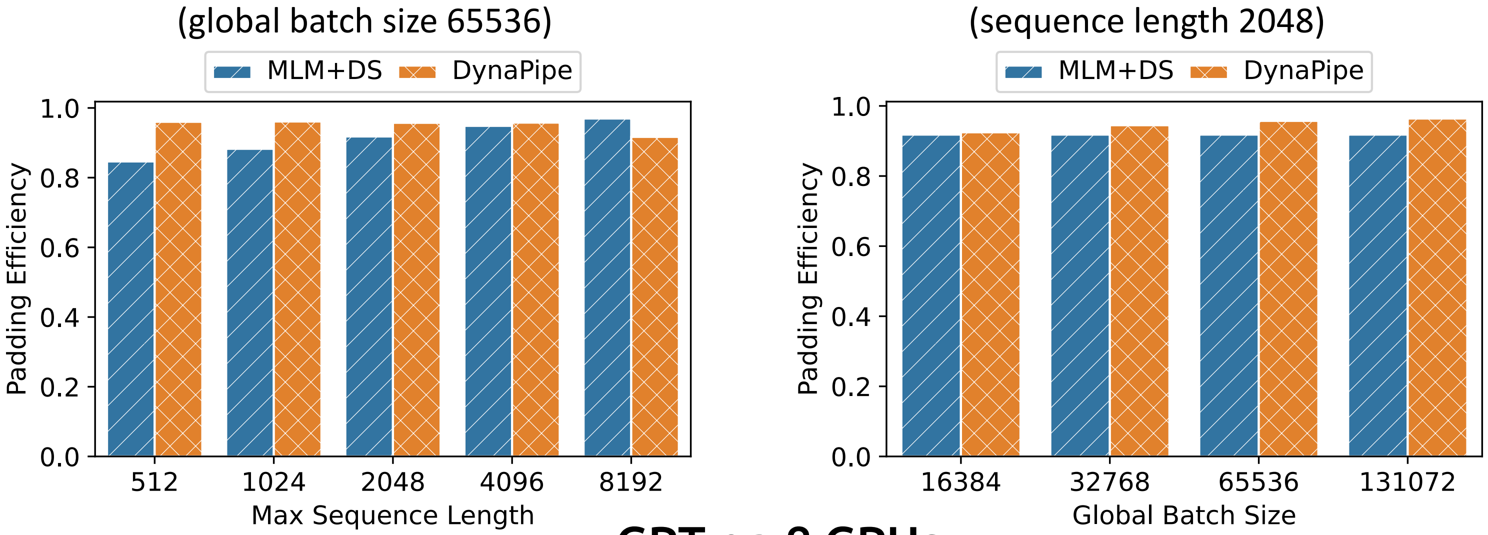


32 A100s

Evaluation

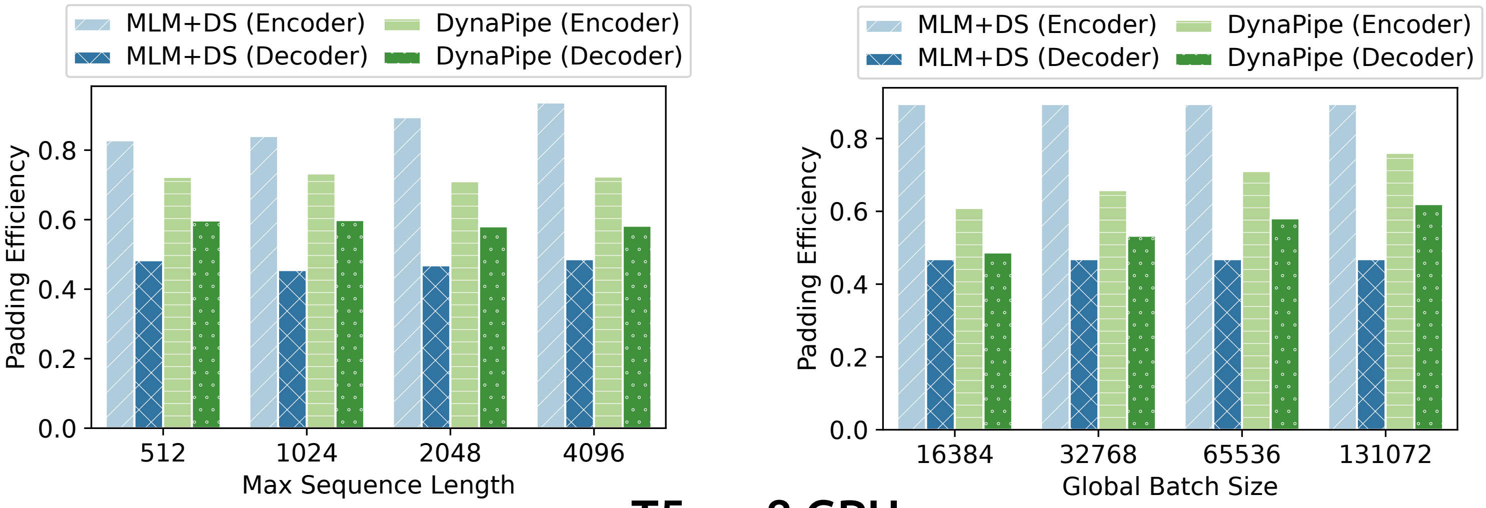
Padding Efficiency

Comparable padding efficiency as packing for GPT.



GPT on 8 GPUs

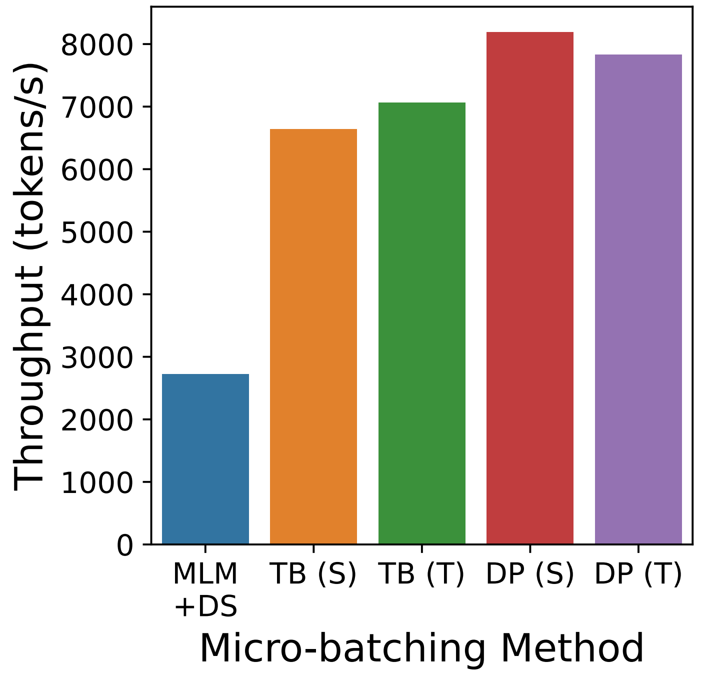
Higher efficiency for T5 decoder, lower for encoder (more balanced overall)

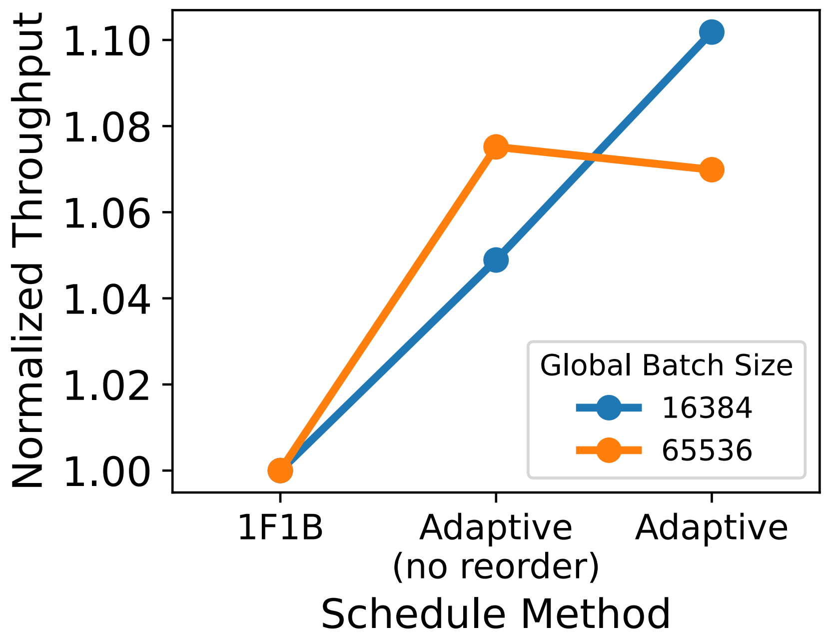


T5 on 8 GPUs

Evaluation

Ablation Study



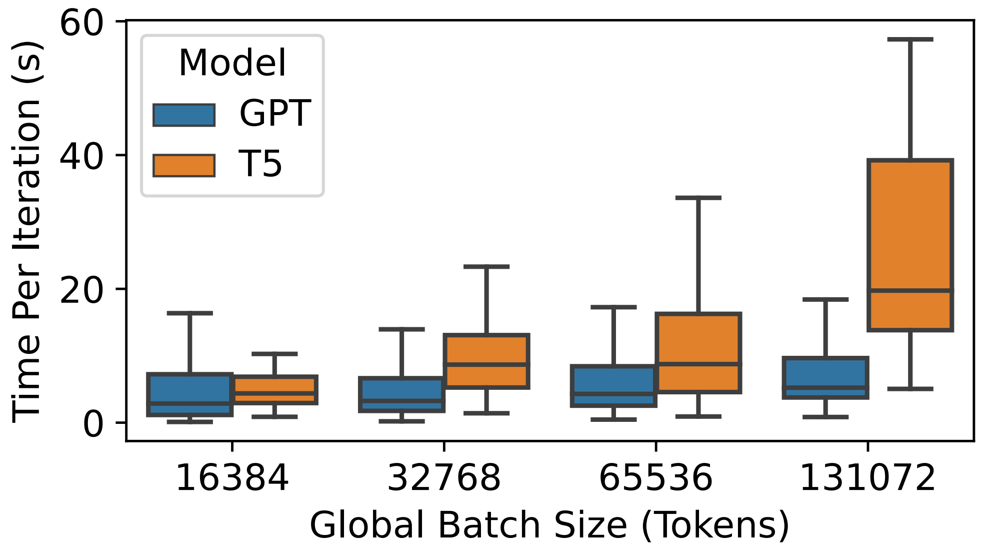


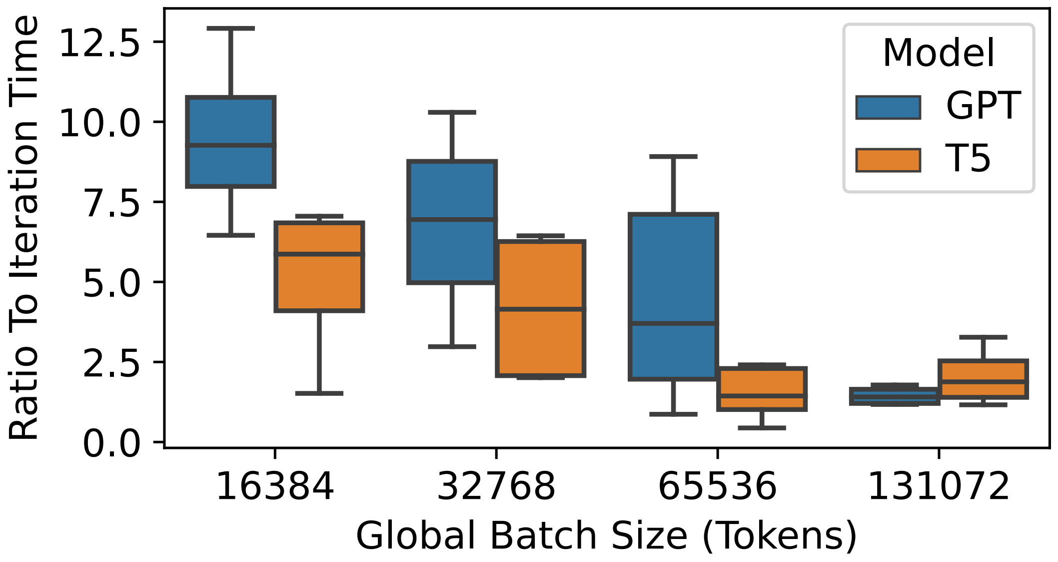
DP algorithm out-performs best token-based micro-batching methods.

Adaptive scheduling out-performs 1F1B.

Evaluation

Planning Time

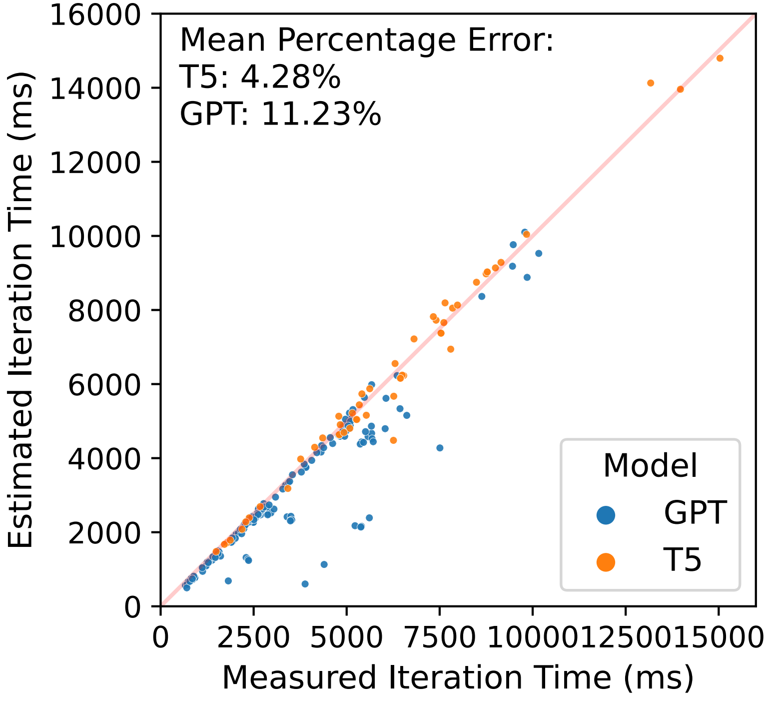


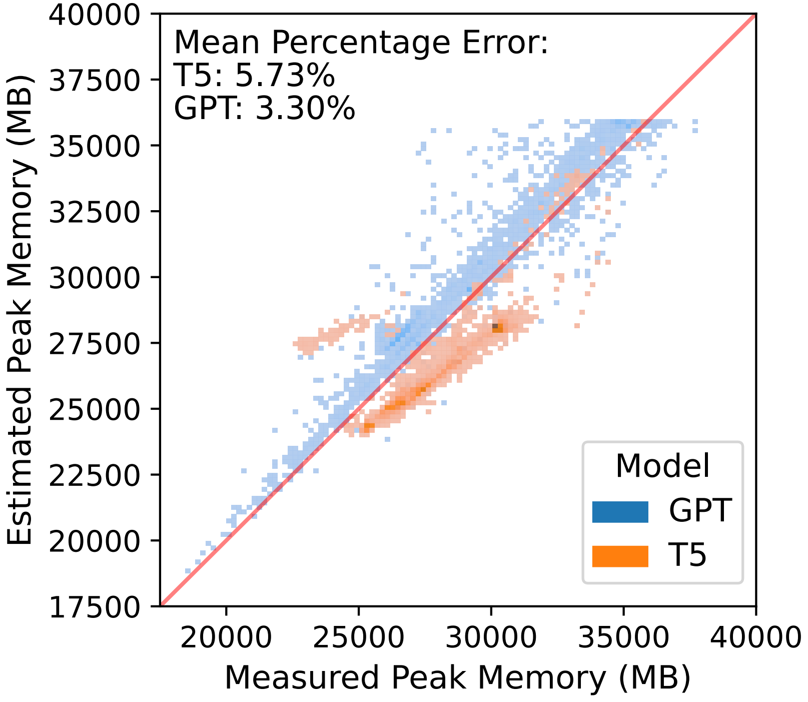


Planning and model training can fully overlap when parallelized to more than 13 CPU cores in all our experiments.

Evaluation

Cost Model Accuracy





Both execution time and memory consumption modelling are accurate, providing useful signal for optimization.