ArrayPipe: Introducing Job-Array Pipeline Parallelism for High Throughput Model Exploration

Presenter: Hongliang Li

Hairui Zhao[1], Hongliang Li[1,2], Qi Tian[1], Jie Wu[3], Meng Zhang[1], Zhewen Xu[1], Xiang Li[1], Haixiao Xu[4]

1 College of Computer Science and Technology, Jilin University, Qianjin street 2699, Changchun, 130012, China

2 Key Laboratory of Symbolic Computation and Knowledge Engineering of the Ministry of Education, Changchun, China

3 Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA

4 High Performance Computing Center, Jilin University, China

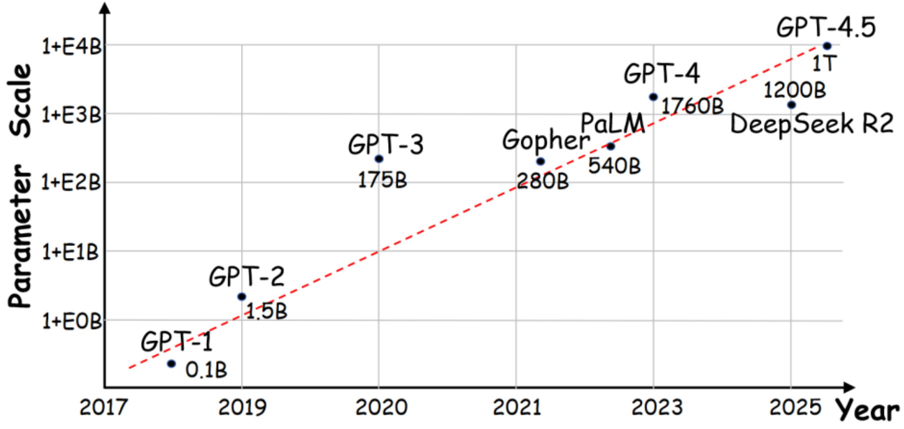
INFOCOM, London, UK, May 2025

Outline

- Background and Motivation
- Job Array Pipeline Parallelism and ArrayPipe Framework
- Evaluations

Background (1)

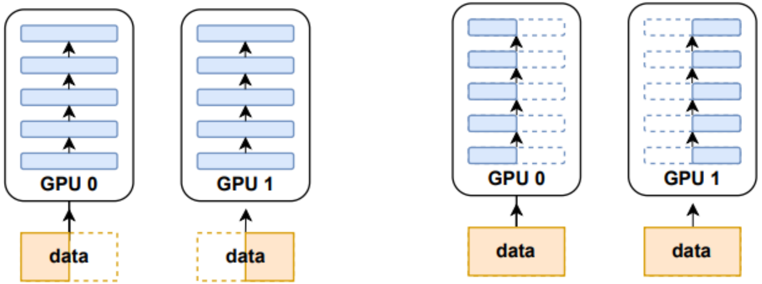
- Parallel schemes to accelerate individual DL model training



Scaling law: bigger model, more data, better result

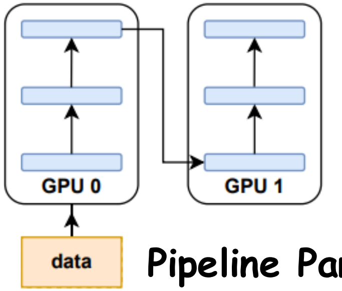
Dramatic growth in both data volume and model complexity.

Training DeepseekV3 requires 2048 H100 GPUs using 55 days.

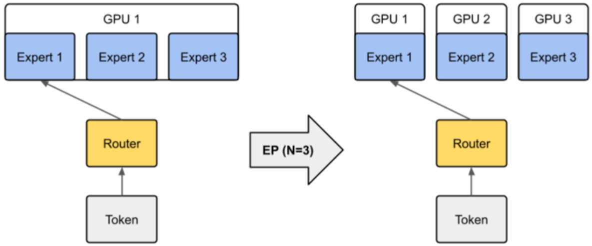


Data Parallel

Tensor Parallel



Pipeline Parallel

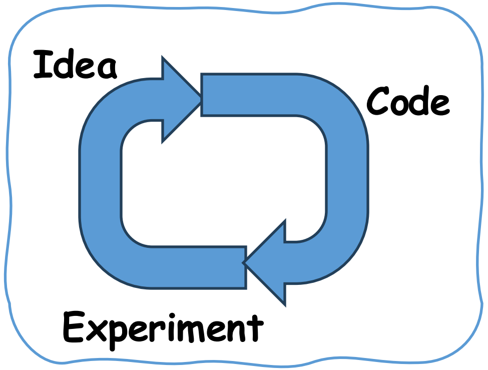


MoE Parallel

[1] Alex Black、Vyacheslav Kokorin. Distributed Deep Learning, Part 1: An Introduction to Distributed Training of Neural Networks

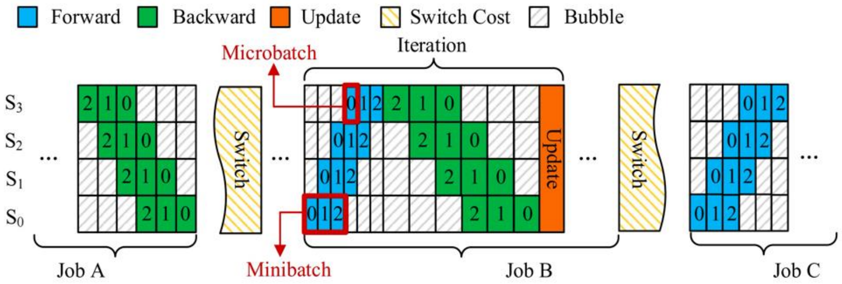
Background (2)

- **Training jobs are not always independent**



Circle of model exploration
(Explore hyperparameter settings)

- The search space for optimal hyperparameter settings can be large (5 hyperparameters with 5 possible values each leads to 3125 configurations).
- Babysitting: trying out one setting at a time (one environment)



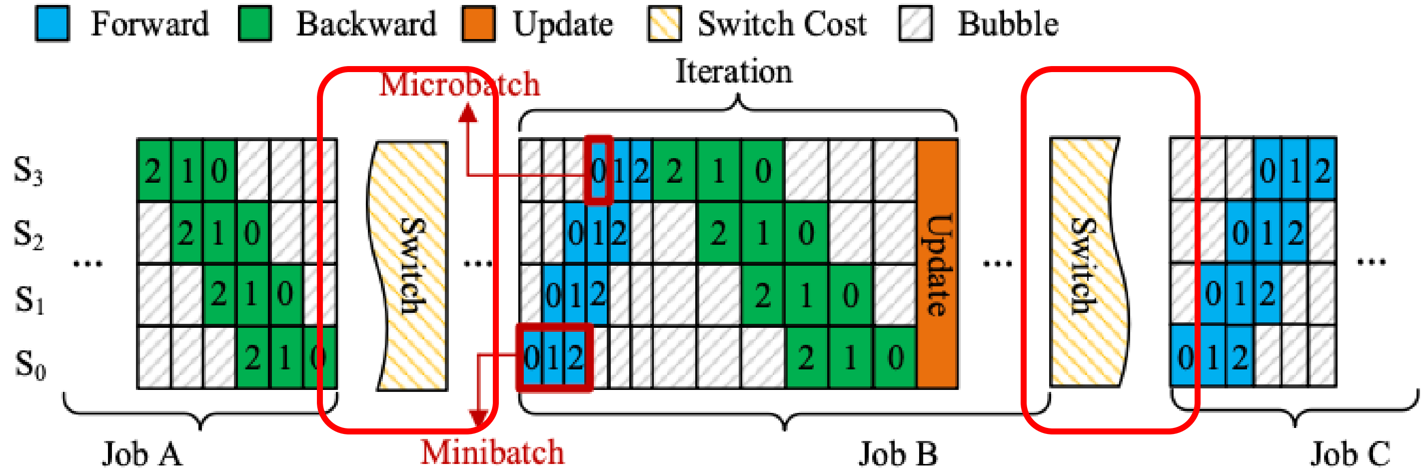
- Batching: Explore multiple settings in parallel (multiple sets of environment)

How to accelerate the model exploration process as a whole?

[1] https://xgboosting.com/optimal-order-for-tuning-xgboost-hyperparameters/?utm_source=chatgpt.com

Motivations (1)

- How to increase the resource utilization in trainings?
  - Reducing "bubbles": divide minibatch to microbatches, pipeline scheduling
  - Overlapping communication with computation
  - Context switching in between jobs (100x the duration of an iteration [1])

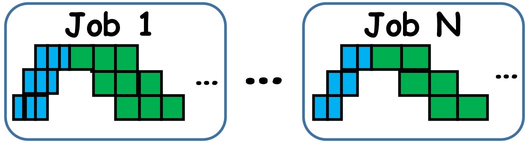


[1] Z. Bai, et al.,"Pipeswitch: Fast pipelined context switching for deep learning applications," in USENIX OSDI, 2020

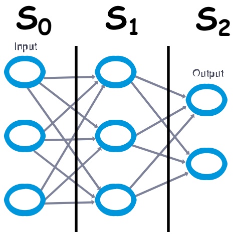
Motivations (2)

- Searching hyperparameter involves trail-and-error sibling jobs
  - share the same core (i.e. model stucture, CUDA Context structure)
  - vary in hyperparameter configuration (e.g., batch_size, learning_rate, weight_decay...)
- Opportunity to pack sibling jobs into a batch (job-array)

Job-Array: a set of Sibling Jobs



Same Model Structure



Different Hyper-Parameter

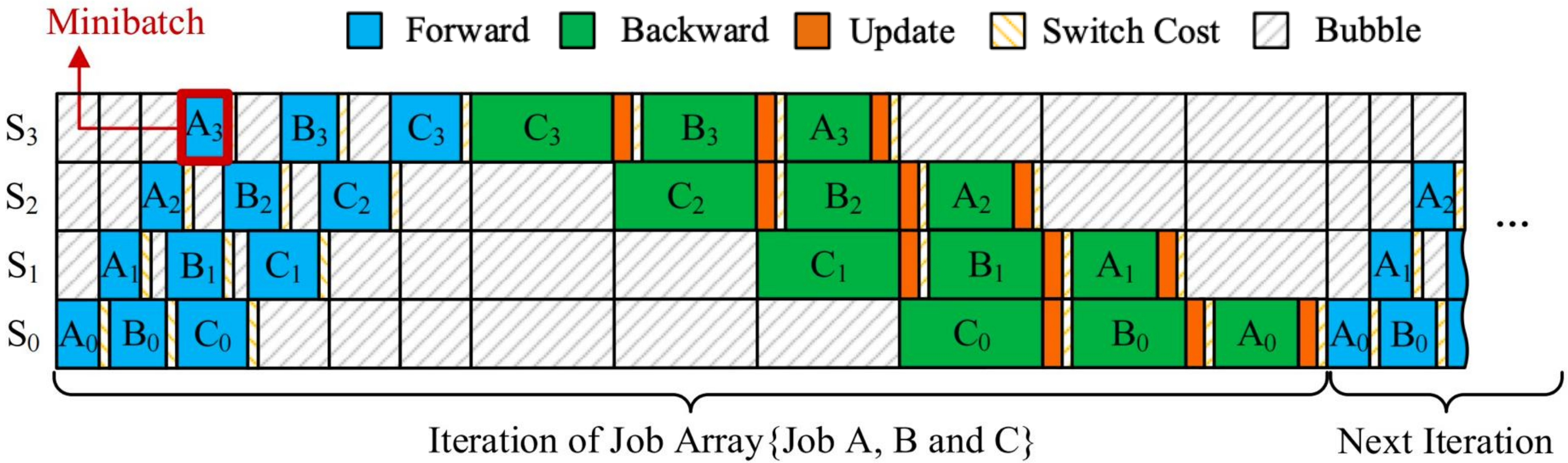
(512, 0.01, 0.0005, …) … (256, 0.01, 0.0005, …)

(batch_size, learning_rate, weight_decay, …)

- Job-Array: train concurrently, instead of sequentially
  i. How to seamlessly switch between sibling jobs?
  ii. How to efficiently schedule multiple jobs in one pipeline?

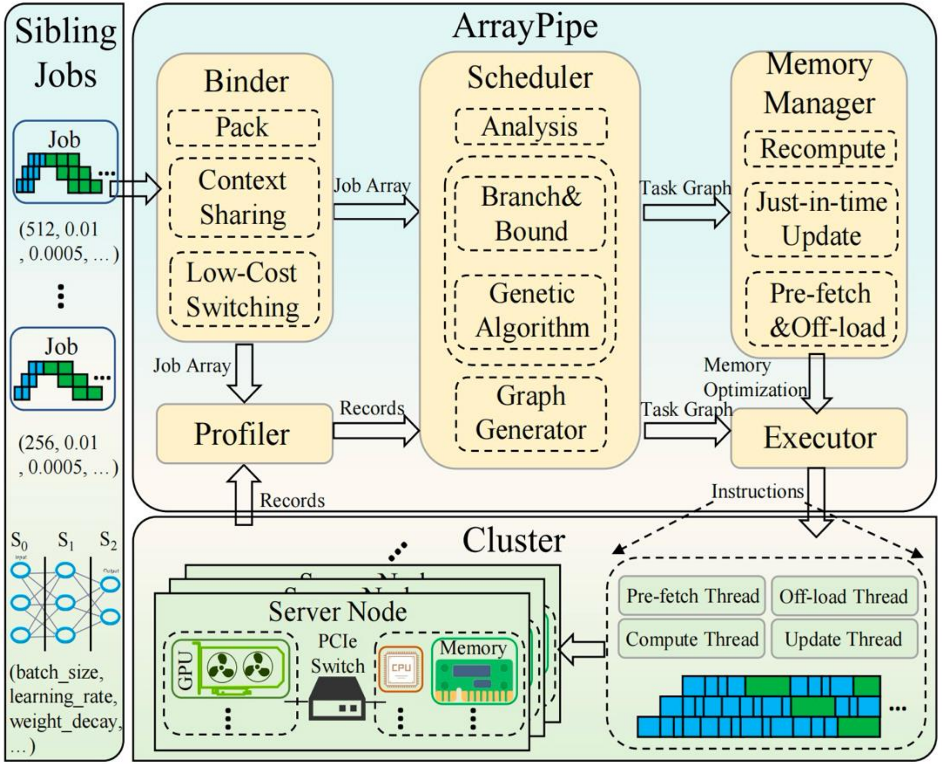
Job-Array Pipeline (JAP) Parallelism

- **Job-level parallelism for high throughput model exploration**



Given a job-array J and a cluster of servers H, ***JAP*** is a pipeline parallelism that supports the stages of different jobs in J execute concurrently rather than consecutively.

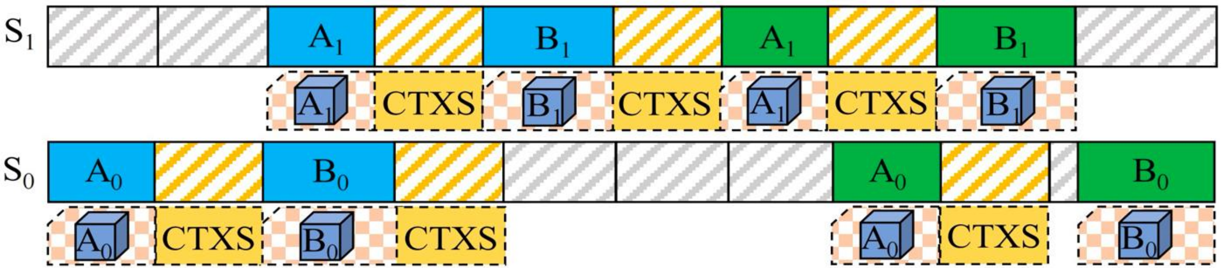
ArrayPipe Framework Overview



- ***Binder*** regards a batch of sibling jobs as a whole and packs them as a job-array supported by **Low-Cost Context Switching (LCS)**.
- ***Scheduler*** integrates two algorithms (B&B and Genetic algorithm) to generate the pipeline plans a job-array.
- ***Memory Manager*** mitigates the memory pressure via re-computation, just in-time updates, memory virtualization, and a dedicated buffer to pre-load model parameters.

Binder

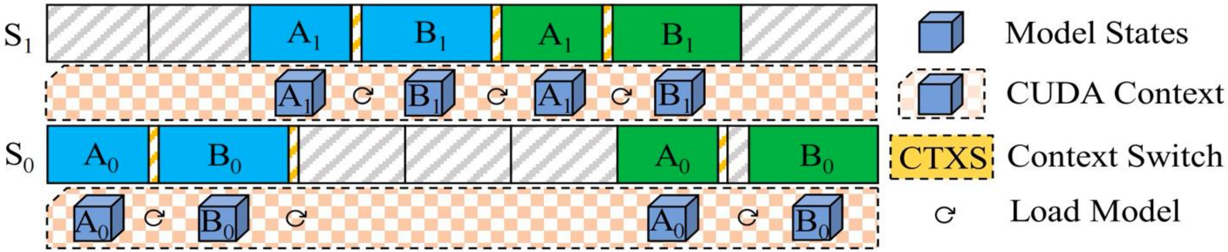
Challenge 1: How to support low-cost context switching between sibling jobs?



(a) Training of *JAP* without sharing CUDA context

Switching context:
- model states
- activations
- gradients

Training with Binder supported by Low-Cost Context Switching (LCS).

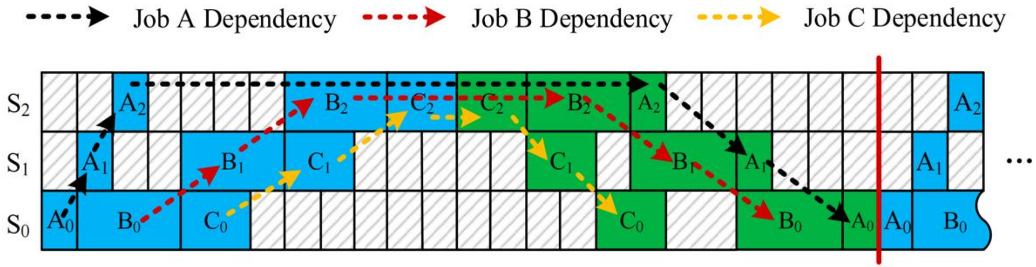


(b) Training of *JAP* with sharing the CUDA context

Switching states:
- parameter
- optimizer

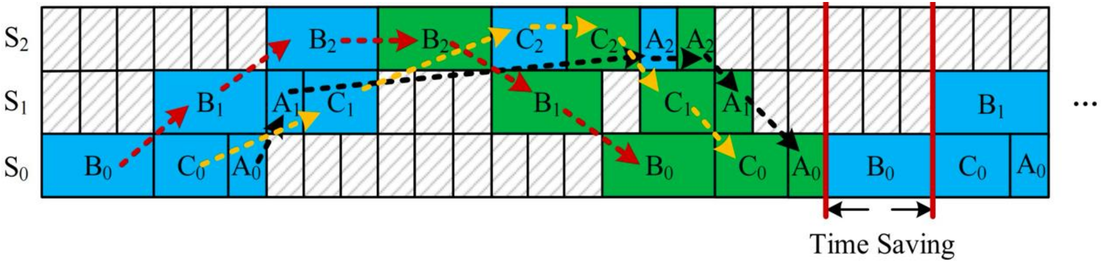
Scheduler

Challenge 2: How to ensure an efficient scheduling when sibling jobs have different mini-batch execution durations and their own stage dependencies?



(a) *JAP* using GPipe Strategy

Sophisticated scheduling algorithms are needed.



(b) ArrayPipe Scheduling Strategy

JAP Scheduling Problem

Given a job-array of DL training jobs J, and a cluster of servers H, the *JAPSP* seeks a schedule of all stages for J on H, with the minimum per-iteration time T, as:

$$minimize \quad T, \tag{4}$$

subject to :

$$e^{j}_{s_i f} + r^{j}_{s_i h f} + \eta^{j}_{s_i f} \leqslant e^{j}_{s_{i+1} f}, \quad i \in \{1, ..., |S| - 1\}, \tag{5}$$

$$e^{j}_{s_i b} + r^{j}_{s_i b} + \eta^{j}_{s_i b} + re^{j}_{s_i hb} \leqslant e^{j}_{s_{i-1} b}, \quad i \in \{2, ..., |S|\}, \tag{6}$$

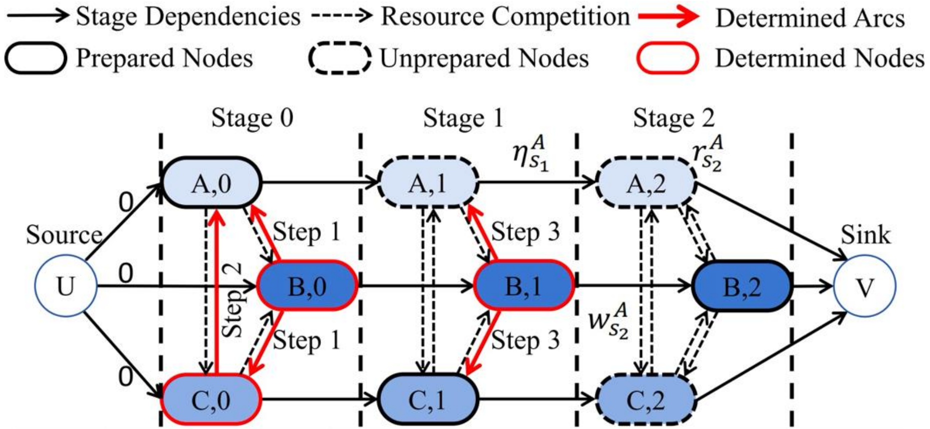
$$e^{j}_{shf} + \eta^{j}_{sf} + w^{j}_{sh} \leqslant e^{j}_{shb}, \quad \forall s \in S, \tag{7}$$

$$e^{j}_{shf} + r^{j}_{shf} + w^{j}_{sh} \leqslant e^{j'}_{shf}, \quad \forall s \in S, \tag{8}$$

$$e^{j}_{shb} + r^{j}_{shb} + w^{j}_{sh} + u^{j}_{sh} + re^{j}_{shb} \leqslant e^{j'}_{shb}, \quad \forall s \in S. \tag{9}$$

Solving JAPSP

- A variant of the Job Shop Scheduling Problem (JSSP), with extra dependencies between model stages
- A Branch-and-bound Algorithm and a Genetic Algorithm



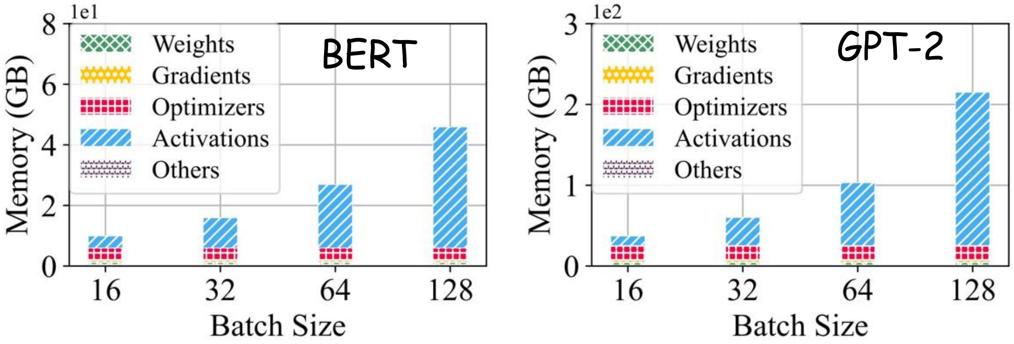
	Ω	$t(\Omega)$	i^*	Ω'	Opt_S
Step1	(A,0),(B,0),(C,0)	$r_{s_0}^A + \eta_{s_0}^A$	Stage 0	(A,0),(B,0),(C,0)	(B,0)
Step2	(A,0),(B,1),(C,0)	$r_{s_0}^A + \eta_{s_0}^A$	Stage 0	(A,0),(C,0)	(C,0)
Step3	(A,0),(B,1),(C,1)	$r_{s_1}^B + \eta_{s_1}^B$	Stage 1	(B,1),(C,1)	(B,1)
...	...	...	...	...	...

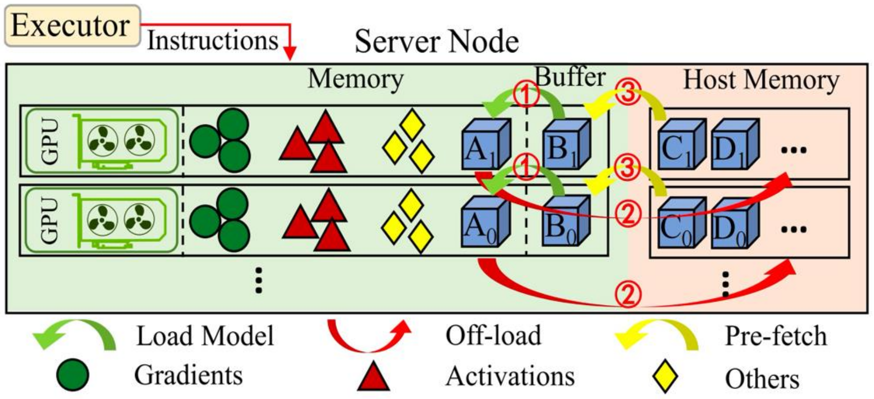
Algorithm 1 Branch and Bound Algorithm for JSSP

Input: Initial graph G, Job set J
Output: Schedule for each *block* of jobs $Sched$
Initial: $LB_{min}(G')=+\infty$, $\Omega'=\{\}$, makespan=0, $Sched=\{\}$, $\Omega=\{(s_1, j)\}$, $Rel_\Omega = \{Rel_{s_1}^j = 0\}, \forall j \in J$
while $\Omega \neq \varnothing$ **do**
  $t(\Omega)=\min_{j\in J}\{Rel_{s_n}^j + r_{s_n}^j\}$
  $(i^*, \_) = \arg\min_{j\in J}\{Rel_{s_n}^j + r_{s_n}^j\}$
  for (s_n, j) in Ω **do**
    if $s_n == i^*$ and $Rel_{s_n}^j < t(\Omega)$ **then**
      $\Omega' = (s_n, j) \cup \Omega'$
    end if
  end for
  for (i^*, j) in Ω' **do**
    $G' = G$
    add arcs to other jobs in stage i^* in G'
    Update makespan
    $L_{max} = 1|Rel_j|Lmax(S)$
    $LB(G')$ = makespan + L_{max}
    if $LB(G') \leq LB_{min}(G')$ **then**
      $Opt_{Sched} = (i^*, j), LB_{min}(G') = LB(G')$
    end if
  end for
  $\Omega = \Omega - Opt\_S \cup (Opt\_S$.next follower)
  Sched = Sched $\cup$ Opt_S
  $G = G'$
end while
return $Sched$

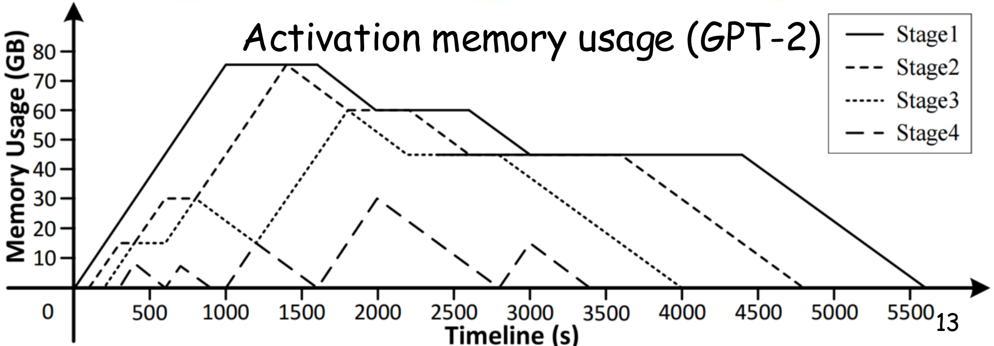
Memory Manager

Challenge 3: How to avoid Out Of Memory (OOM) in JAP when too many sibling jobs are sharing resources?





- Activation account of nearly 60% of the memory footprint
- Re-materialization: Eliminate and recompute



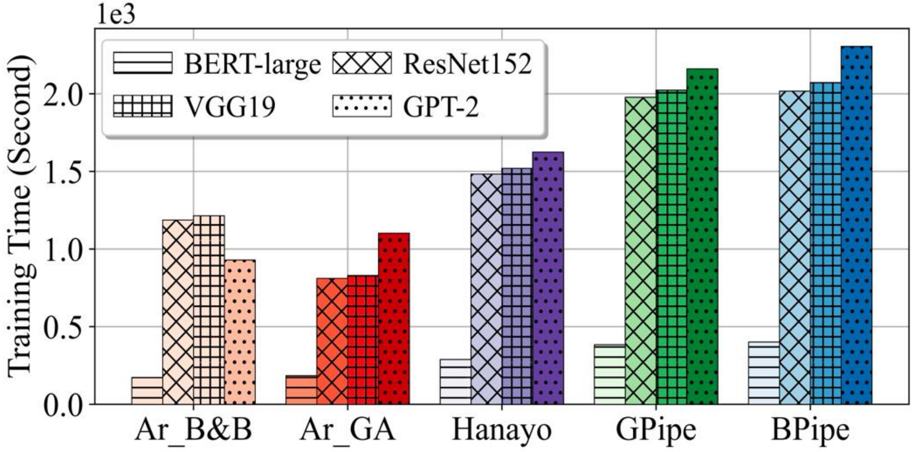
Evaluations

Testbed: NVIDIA SXM4 server with 8 A100 (80GB), NVLink (300GB/s), PCIe4.0 (64GB/s).

Baselines: GPipe, BPipe, Hanayo

DNN Models:

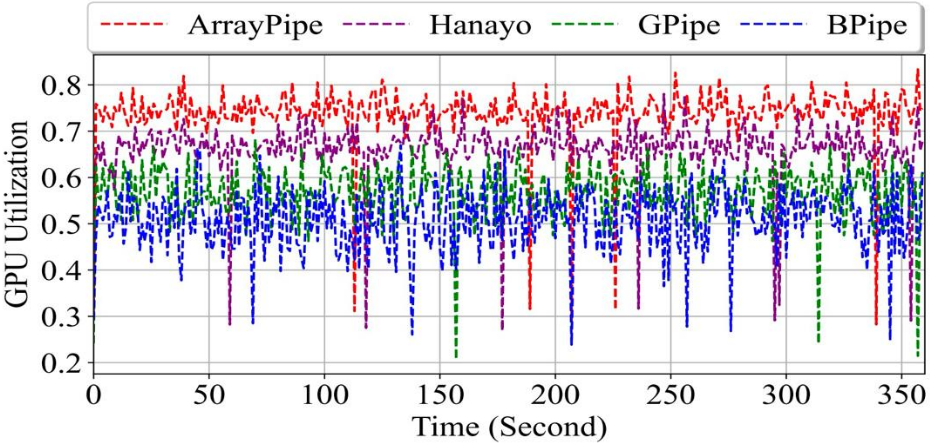
Model	Dataset	Optimizer	# of Params
VGG19	ImageNet	SGD	144M
ResNet152	ImageNet	SGD	60M
BERT-large	GLUE	BertAdam	340M
GPT-2	WikiText	AdamW	1.5B



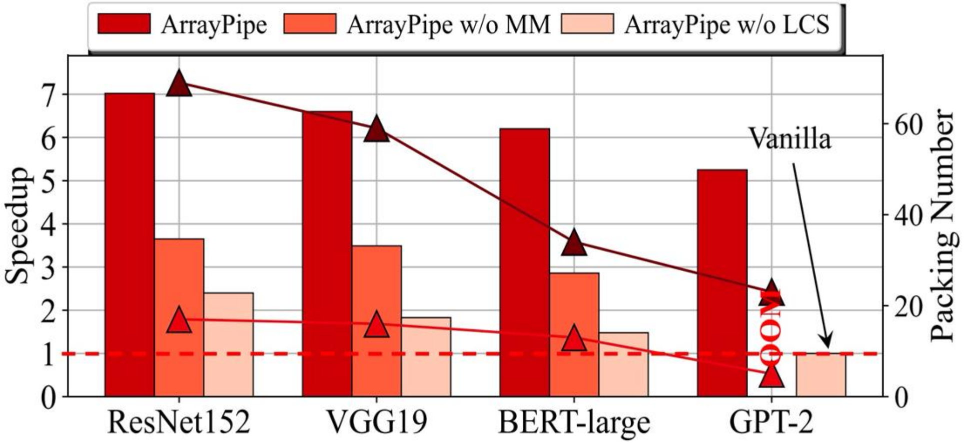
Training Time Comparisons.

- ArrayPipe achieves **1.46×** **training throughput**, comparing with State-Of-The-Art (SOTA) PPs, on average.

Evaluations



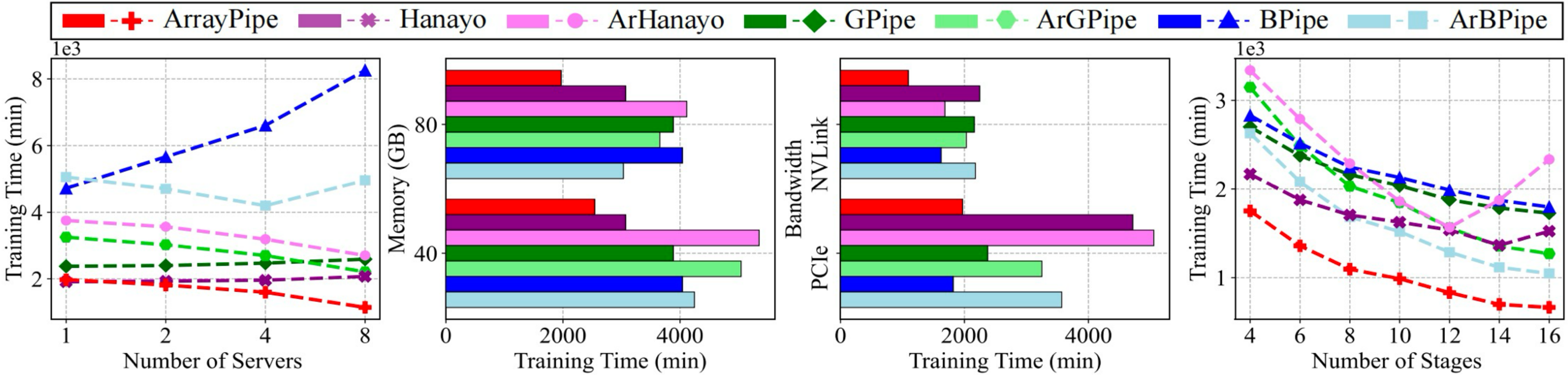
GPU Utilization.



Comparison between whether to use LCS and MM in ArrayPipe.

- ArrayPipe achieves high GPU utilization. As **more jobs** are packed into a job-array, the overall **throughput improves**.

Large-scale Simulation



(a) Performance with Different Number of Servers

(b) Performance with different Memory of GPUs

(c) Performance with Different Bandwidth of GPUs

(d) Performance with Different Number of Stages

- ArrayPipe is more suitable for environments with **larger GPU memory capacity**, **higher intra-node bandwidth**, aligning with the developing trends and application patterns.

Conclusion and Future Work

Conclusion

i. A novel parallel scheme (JAP) is introduced to enable a batch of sibling jobs to form a concurrent job-array and to execute concurrently, targeting high throughput model exploration.

ii. We design ArrayPipe, a framework to support JAP with low-cost job context switching within a job-array and a GPU-Host memory manager for higher training concurrency.

iii. We propose a novel scheduling problem JAPSP that seeks to minimize the per-iteration time of a job-array, along with two algorithms for different scales of job-arrays.

iv. Extensive testbed experiments and trace-driven simulations are conducted to evaluate the efficiency of ArrayPipe.

Future Work

i. How can ArrayPipe handle exploratory job workflows that frequently terminate and resubmit?

ii. Models with different hyperparameters may benefit from different degrees of parallelism. Fine-grained (layer-wise) LCS and migration is a possible direction.

Thank you

Presenter: Hongliang Li

lihongliang@jlu.edu.cn
College of Computer Science and Technology,
Jilin University, Changchun, China, 130012