DiffusionPipe: Training Large Diffusion Models with Efficient Pipelines

Ye Tian, Zhen Jia, Ziyue Luo, Yida Wang, Chuan Wu

The University of Hong Kong

Amazon Web Services

Diffusion model taxonomy

During training, a diffusion model typically comprises:

1. Trainable components, e.g., U-Net / transformer backbone
2. Non-trainable components, e.g., frozen input encoders

Executing non-trainable components takes a significant proportion of time in diffusion model training.

Non-trainable part execution time / (forward + backward time of the trainable part)

Model / Batch size	**8**	**16**	**32**	**64**
Stable Diffusion v2.1	38%	41%	43%	44%
ControlNet v1.0	76%	81%	86%	89%

Data parallelism

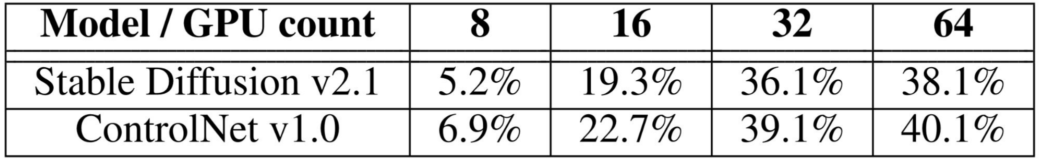
Currently most applied training scheme to diffusion models.

Consume much memory and restricts training batch size.

For example, Stable-Diffusion v2.1 consumes 24.3 GB at local batch size 8, while TPUv3 (32 GB) is used in its original paper.

Involve significant synchronization overhead when the cluster is large.

Synchronization time / iteration training time at local batch size 8



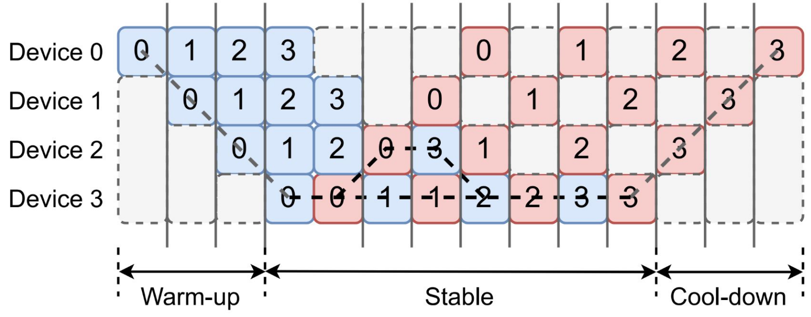
Model / GPU count	**8**	**16**	**32**	**64**
Stable Diffusion v2.1	5.2%	19.3%	36.1%	38.1%
ControlNet v1.0	6.9%	22.7%	39.1%	40.1%

Pipeline parallelism

Less synchronization overhead and memory consumption, as a device is only responsible for part of the model.

Support larger training batch size and potentially better device utilization.

Example of FIFO-1F1B pipeline schedule

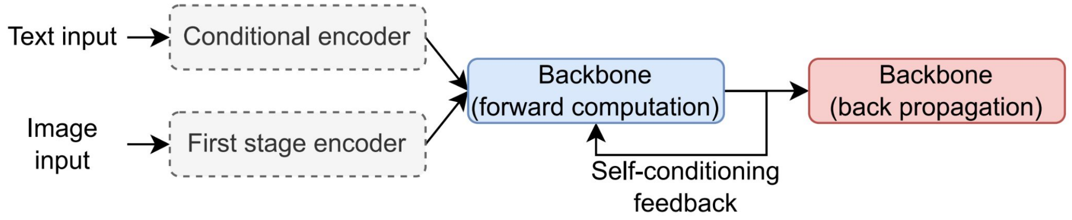


Challenges (characteristics of diffusion models)

Structural characteristics and special training procedures of diffusion models:

1. Existing methods cannot readily handle non-trainable part and multiple (independent) backbone models.
2. Popular self-conditioning training technique involves an additional forward stage for the trainable part.

Training procedure of Stable-Diffusion with self-conditioning


Text input
Conditional encoder
Image input
First stage encoder
Backbone (forward computation)
Backbone (back propagation)
Self-conditioning feedback

Challenges (pipeline bubble)

Significant pipeline bubbles in synchronous pipeline training.

Unique opportunity: filling pipeline bubbles with the non-trainable part.

To handle:

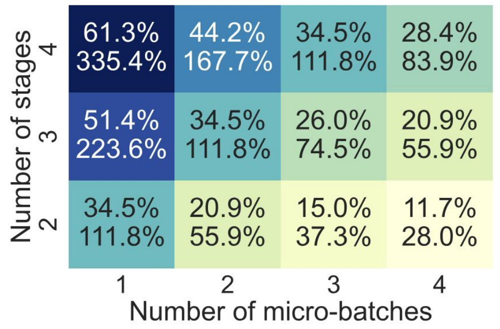
1. Data dependency between trainable and non-trainable part.
2. Layer with extra-long execution time that may not fit into any pipeline bubble.

Pipeline bubble filling

Non-trainable part:

1. Execution not restricted by synchronization barrier in pipeline parallel training.
2. Execution time comparable to pipeline bubble time.

Lower row: non-trainable part execution time / pipeline bubble time

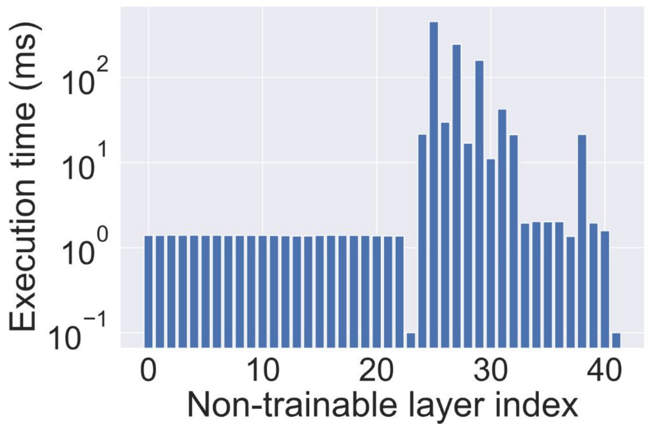


Number of stages \ Number of micro-batches	1	2	3	4
4	61.3% 335.4%	44.2% 167.7%	34.5% 111.8%	28.4% 83.9%
3	51.4% 223.6%	34.5% 111.8%	26.0% 74.5%	20.9% 55.9%
2	34.5% 111.8%	20.9% 55.9%	15.0% 37.3%	11.7% 28.0%

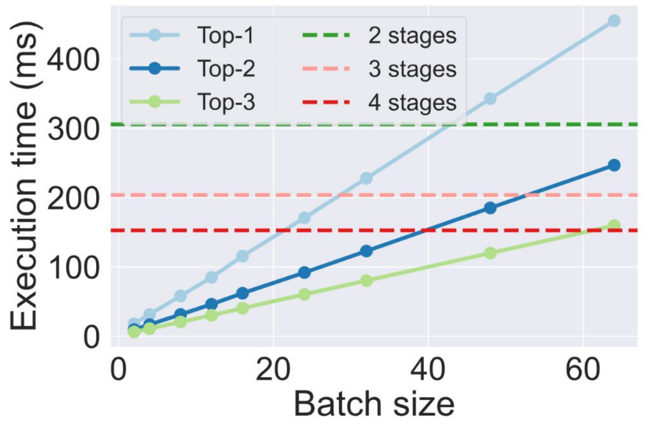
Extra-long non-trainable layer

Extra-long layer's execution may last hundreds of milliseconds.

Execution time of non-trainable layers at batch size 64 of Stable-Diffusion v2.1



Execution time of top-3 longest non-trainable layer and longest pipeline bubble time at 4 micro-batches

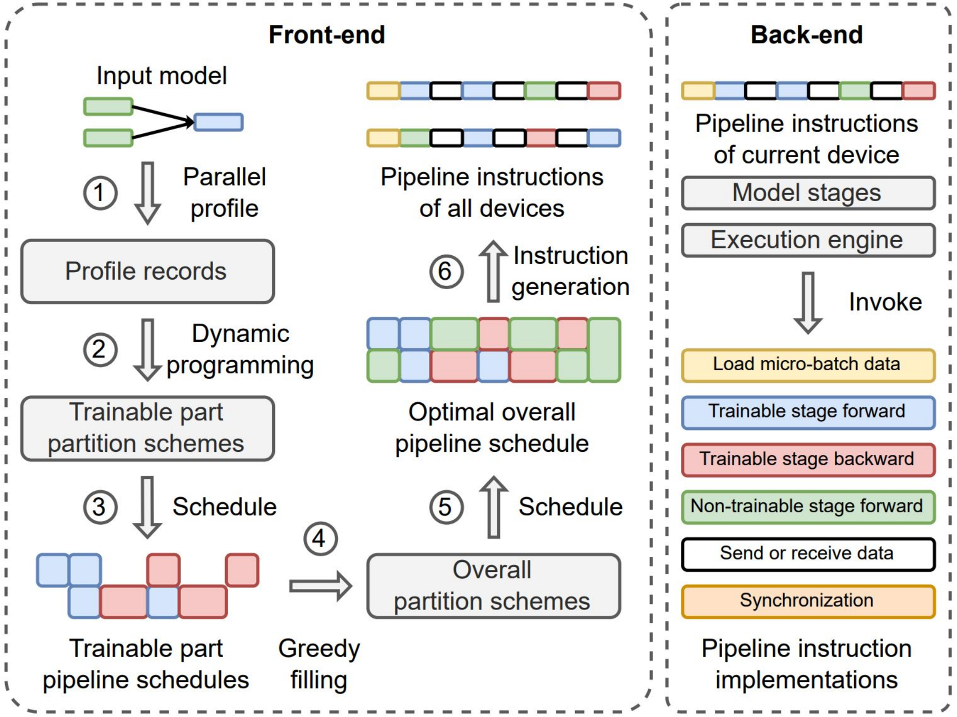


DiffusionPipe: system design

Front-end: generates an optimized pipeline training schedule:

1. Model partitioning scheme of the trainable part.
2. Bubble filling scheme of the non-trainable part.

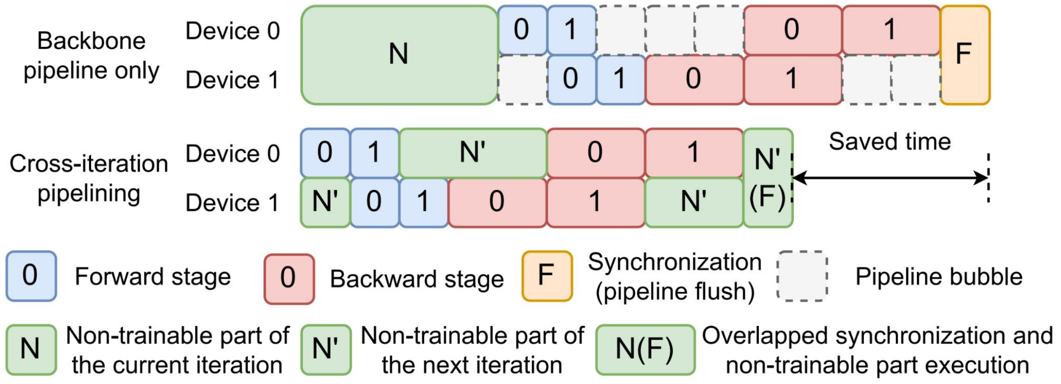
Back-end: performs training according to pipeline schedule.



Pipeline schedule: cross-iteration pipelining

Overlapping the backbone pipeline training in an iteration with the non-trainable part execution of the next iteration.

Solution to challenge: data dependency between trainable and non-trainable part blocks pipeline bubble filling.



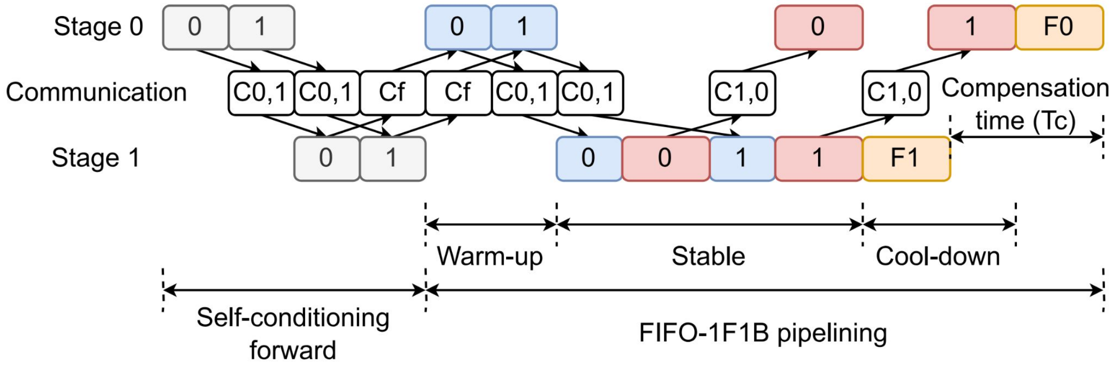
Backbone pipeline only
Device 0
Device 1
N
0
1
F
Cross-iteration pipelining
Device 0
Device 1
N'
N' (F)
Saved time
Forward stage
Backward stage
Synchronization (pipeline flush)
Pipeline bubble
Non-trainable part of the current iteration
Non-trainable part of the next iteration
N(F)
Overlapped synchronization and non-trainable part execution

Model partitioning

Cases: single backbone, multiple backbones, self-conditioning.

Input: number of pipeline stages and devices in a pipeline parallel group.

Algorithm: dynamic programming (DP), optimizing iteration training time.



Formulation: single backbone

Optimization target: max pipeline computation time + max net synchronization time (i.e., synchronization time - compensation time).

$$\min_{1\leq r\leq D}\{(M+2S-2)\boxed{W(L,S,r,D)}+\boxed{Y(L,S,r,D)}\}$$

$$T_0(s)=\max\{\sum_{l<i\leq L}\mathbf{P}_i^f(\frac{B}{r})+\sum_{l<i\leq L}\mathbf{P}_i^b(\frac{B}{r}),\ \frac{\mathbf{C}^f_{l,l+1}(\frac{B}{r})+\mathbf{C}^b_{l+1,l}(\frac{B}{r})}{\mathbf{R}_{p2p}}+2\mathbf{L}_{p2p}\}$$

$$\boxed{T_S(s)=\sum_{l<i\leq L}\mathbf{G}_i(\frac{B}{r})/\mathbf{R}_{ar}+\mathbf{L}_{ar}}$$

$$\boxed{T_C(s)=\sum_{l<i\leq L}\mathbf{P}_i^b(\frac{B}{r})}$$

$$T_0^{S-C}(s)=T_S(s)-T_C(s)$$

$$W(L,S,r,D)=\max\{W(l,S-1,r',D-r),T_0(s)\}$$

$$Y(L,S,r,D)=\max\{Y(l,S-1,r',D-r),T_0^{S-C}(s)\}$$

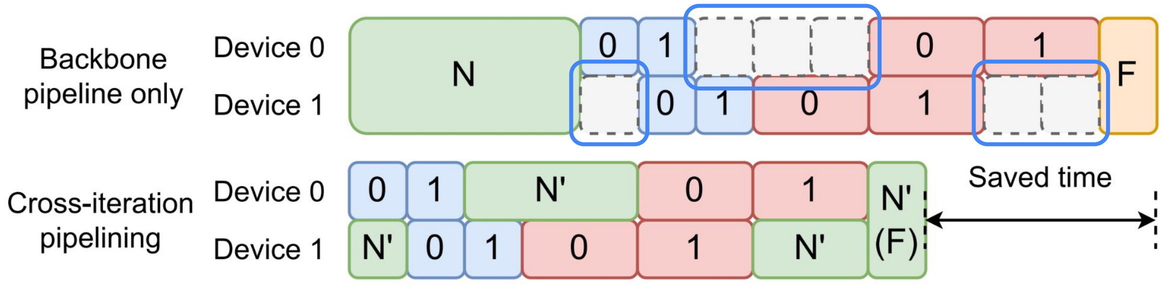
$$\forall l,r',\quad 1\leq l\leq L-1,\ 1\leq r'\leq D-r$$

Pipeline bubble filling procedure

Pipeline bubble: (start time, end time, idle devices).

Identify all idle devices at the same idle period as a pipeline bubble.

Fill pipeline bubbles one by one in chronological order.

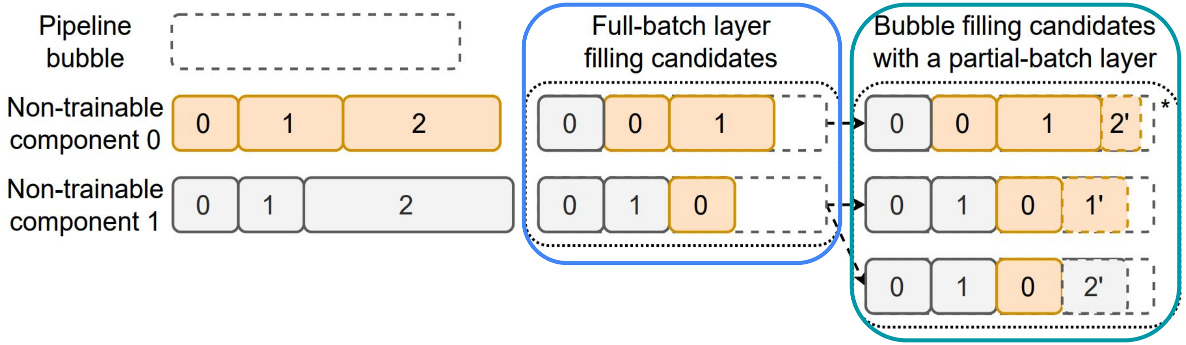


Algorithm: filling one pipeline bubble

Greedily generate all possible bubble filling candidates with full-batch layers.

Try inserting only one partial-batch layer to each full-batch layer filling candidate.

Select the candidate with longest execution time as the final bubble filling scheme.


Pipeline bubble
Non-trainable component 0
0
1
2
Non-trainable component 1
0
1
2
Full-batch layer filling candidates
0
0
1
0
1
0
Bubble filling candidates with a partial-batch layer
0
0
1
2'
*
0
1
0
1'
0
1
0
2'

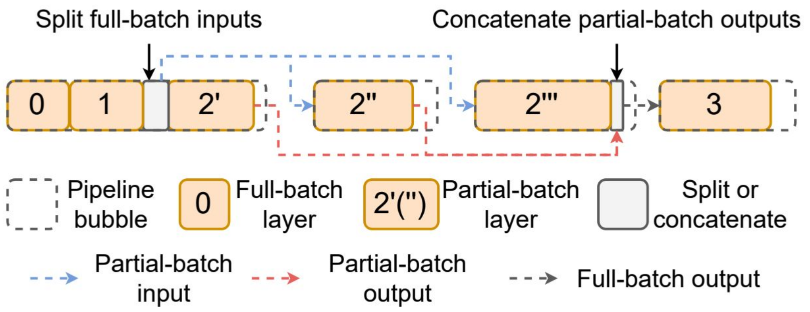
Partial-batch layer

Partial-batch layer: (component index, layer index, number of samples in partial-batch).

Motivation: reduce bubble filling granularity and handle extra-long non-trainable layers.

Restriction: full-batch layers + partial-batch layer execution time <= idle time.

Implementation

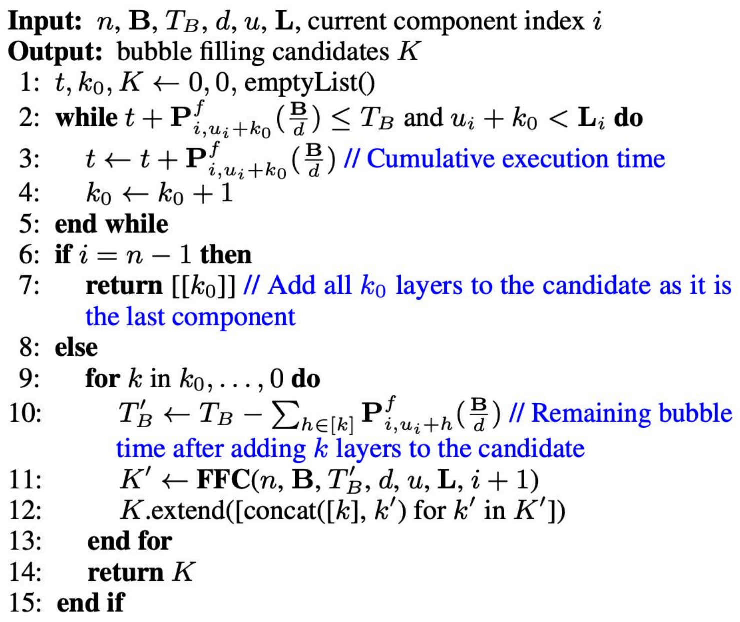


Algorithm: greedily finding full-batch layer candidates

Iterate on available non-trainable components.

Compute maximum number of layers of the current non-trainable component that can be inserted.

Insert certain number of layers into the bubble; then try to insert the next non-trainable component by recursively calling itself.



```
Input: $n$, $\mathbf{B}$, $T_B$, $d$, $u$, $\mathbf{L}$, current component index $i$
Output: bubble filling candidates $K$
$t, k_0, K \leftarrow 0, 0,$ emptyList()
while $t + \mathbf{P}^f_{i,u_i+k_0}(\frac{\mathbf{B}}{d}) \leq T_B$ and $u_i + k_0 < \mathbf{L}_i$ do
    $t \leftarrow t + \mathbf{P}^f_{i,u_i+k_0}(\frac{\mathbf{B}}{d})$ // Cumulative execution time
    $k_0 \leftarrow k_0 + 1$
end while
if $i = n - 1$ then
    return $[[k_0]]$ // Add all $k_0$ layers to the candidate as it is the last component
else
    for $k$ in $k_0, \ldots, 0$ do
        $T'_B \leftarrow T_B - \sum_{h\in[k]} \mathbf{P}^f_{i,u_i+h}(\frac{\mathbf{B}}{d})$ // Remaining bubble time after adding $k$ layers to the candidate
        $K' \leftarrow$ FFC($n$, $\mathbf{B}$, $T'_B$, $d$, $u$, $\mathbf{L}$, $i+1$)
        $K$.extend([concat([$k$], $k'$) for $k'$ in $K'$])
    end for
    return $K$
end if
```

Experiment setup

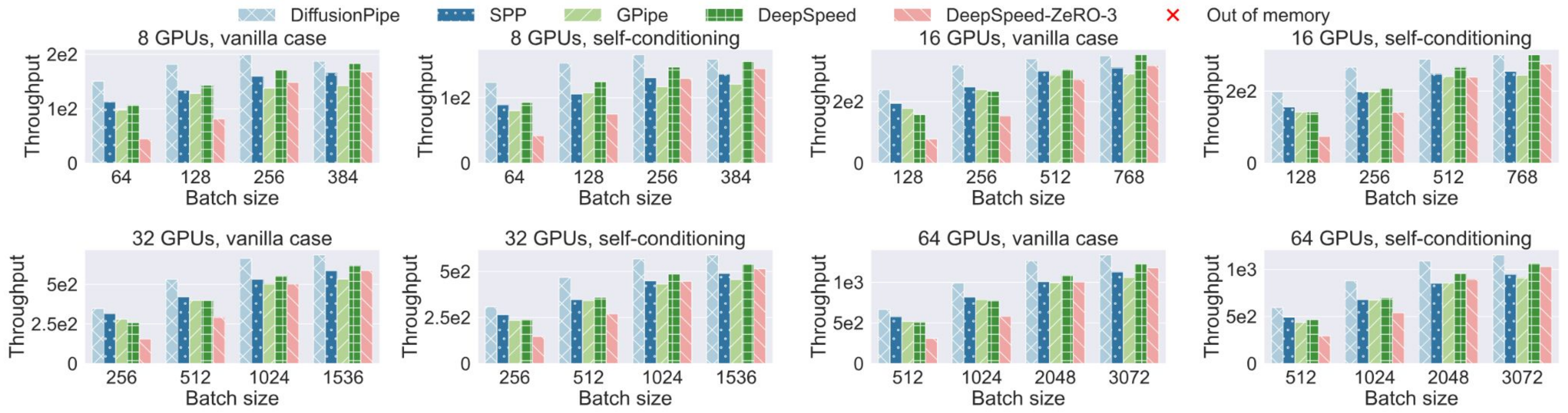
Test-bed: 8 Amazon EC2 p4de.24xlarge (64 A100-80GB GPUs).

Baselines: DeepSpeed (w/ and w/o ZeRO-3), GPipe, SPP.

Metric: throughput (samples / second).

Model	**Input shape**	**Self-conditioning**
Stable-Diffusion v2.1	512x512	Enabled
ControlNet v1.0	512x512	Enabled
CDM-LSUN	64x64, 128x128	Not enabled
CDM-ImageNet	64x64, 128x128	Not enabled

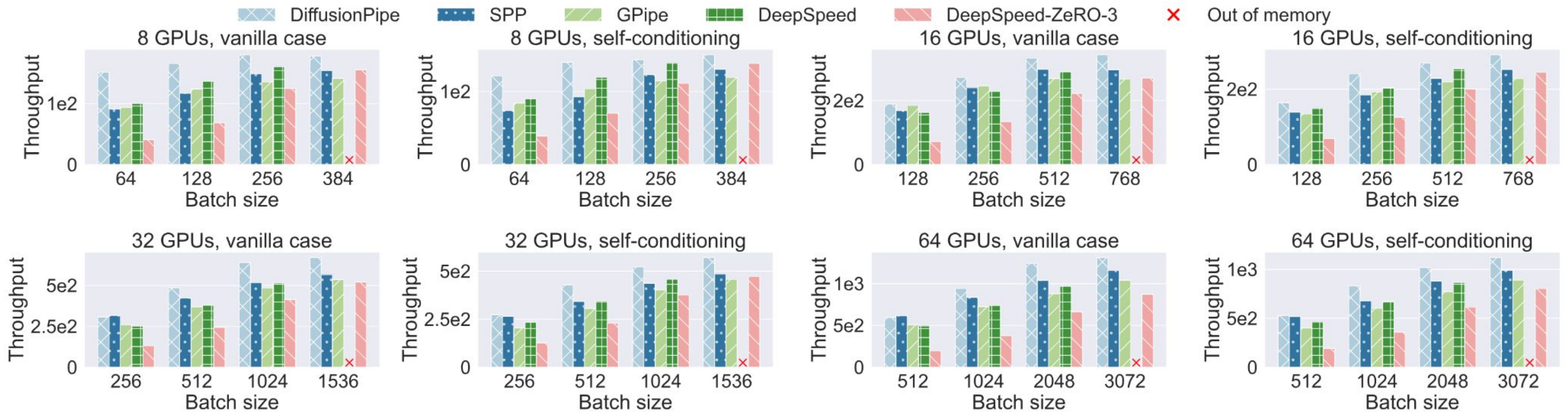
Throughput: Stable-Diffusion v2.1



Outperform other pipeline parallel baselines due to better pipelining efficiency.

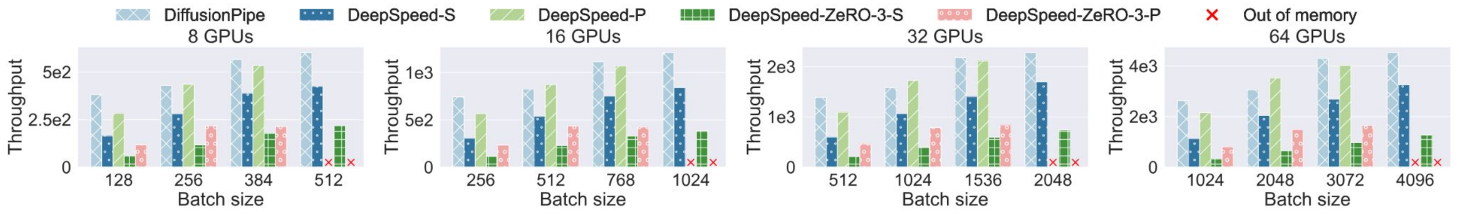
Slightly outperform data parallel baselines due to higher device utilization brought by larger local batch size.

Throughput: ControlNet v1.0

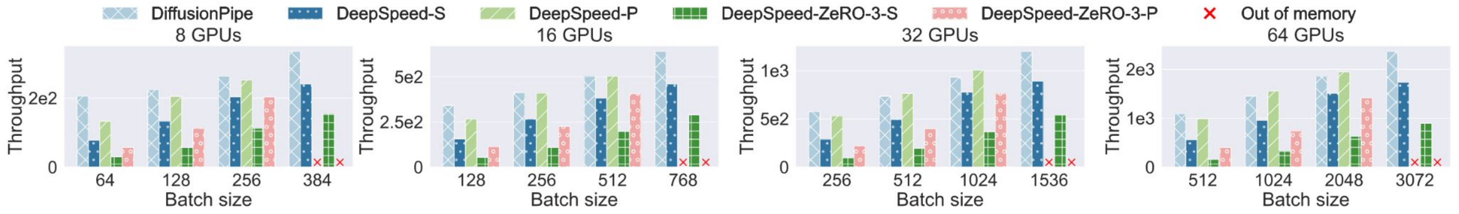


Outperform pipeline parallel schemes and even better than data parallelism.

Throughput: CDM models

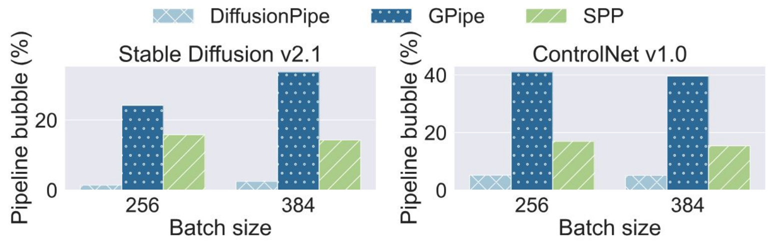


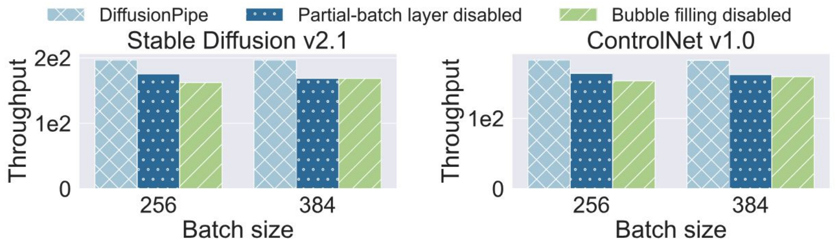
(c) CDM-LSUN



Limited speedup due to few non-trainable parts.

Pipeline bubble ratio & ablation study





Pipeline bubble ratio: sum of (pipeline bubble time * number of idle devices) / (iteration time * total number of devices).

Pipeline bubble is significantly reduced by applying bubble filling.

Disabling partial-batch layer significantly degrades throughput.

Disabling bubble filling degrades it even more.

Summary & takeaway

Summary:

1. DiffusionPipe achieves higher training throughput compared with baselines.
2. Pipeline bubble filling significantly reduces bubble ratio.
3. Partial-batch layer design effectively improves bubble filling efficiency.

Takeaway:

1. Our design can be applied to other models, e.g., diffusion models with transformer backbones and multimodal models.
2. Our design can be extended to dynamic workload, e.g., variable length inputs.

Thanks!

We will open source soon!
Contact: yetiansh@connect.hku.hk