FlexGen

High-throughput Generative Inference of Large Language Models with a Single GPU

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E. Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, Ce Zhang

Presented by Steven Gao

Outline

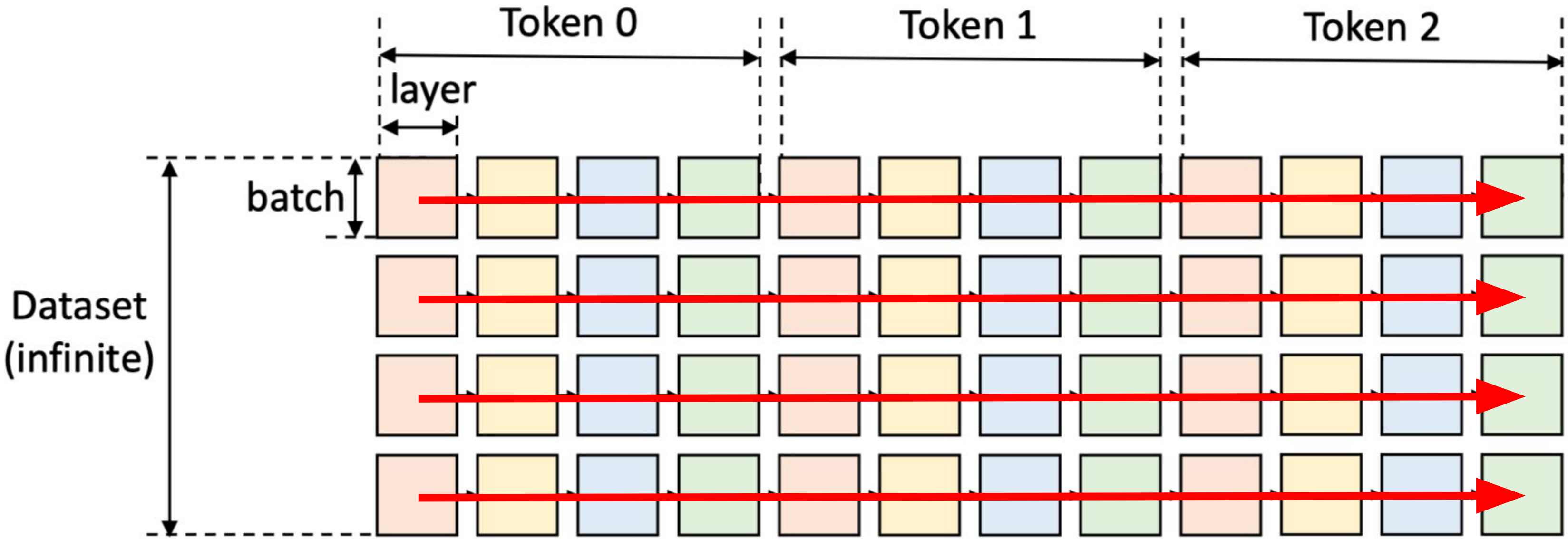
- Challenges in Inference
- Motivation
- Block Schedule
- Cost Model
- Compression
- Evaluation

From Training to Inference

- Latency
  - Small batch sizes
  - Memory bound
  - Maximize memory bandwidth utilization when reading weights for feed-forward layers
- Decoding throughput
  - Overlapping compute with reading model weights
    - We can increase throughput if latency remains the same per layer
    - Maximize batch size
  - Autoregressive decoding
    - Dependency on previous token (doesn’t exist in training)
    - Need to maintain KV cache longer
    - Higher memory requirements

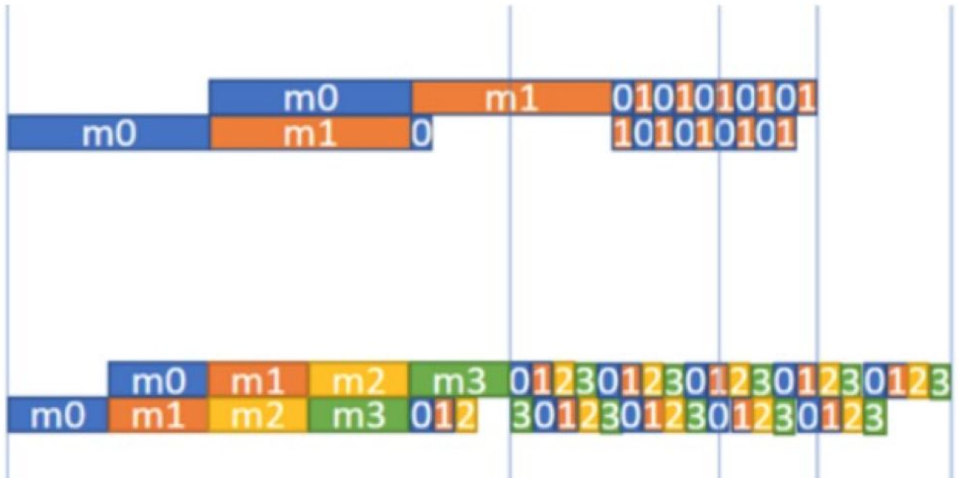
Latency-Optimal Schedule

- Previous works go row by row in the compute graph



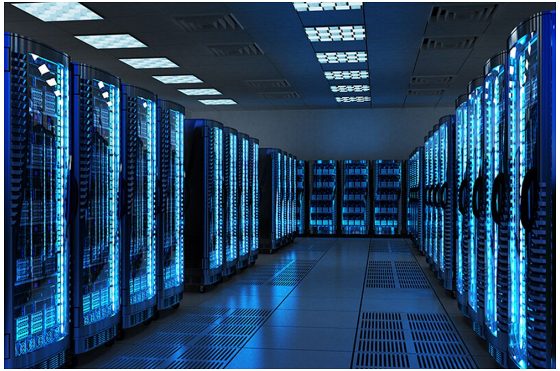
Latency-Optimal Schedule

- Previous works go row by row in the compute graph
  - Low latency



Motivation

- Make large models more accessible



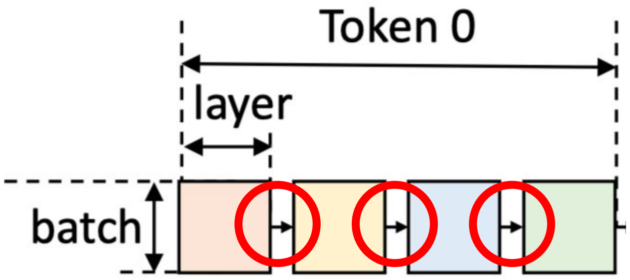
Motivation

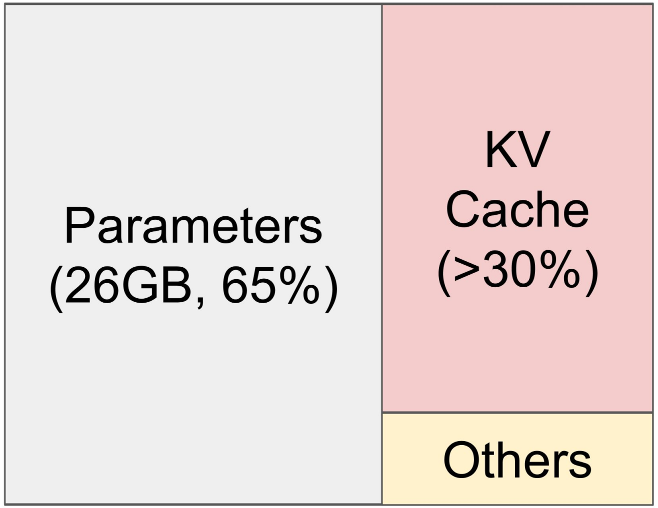
Latency oriented tasks
(e.g. chatbots)

Throughput oriented tasks
(e.g. offline document processing)

Throughput-Optimal Schedule

- Previous works go row by row in the compute graph
  - Low latency
  - High IO

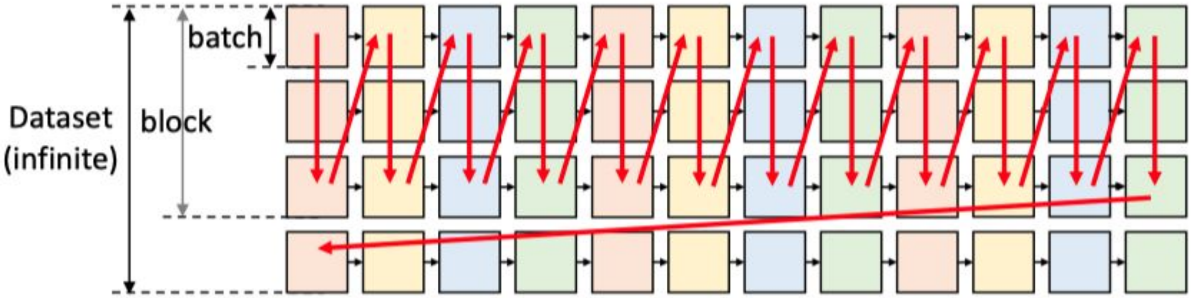




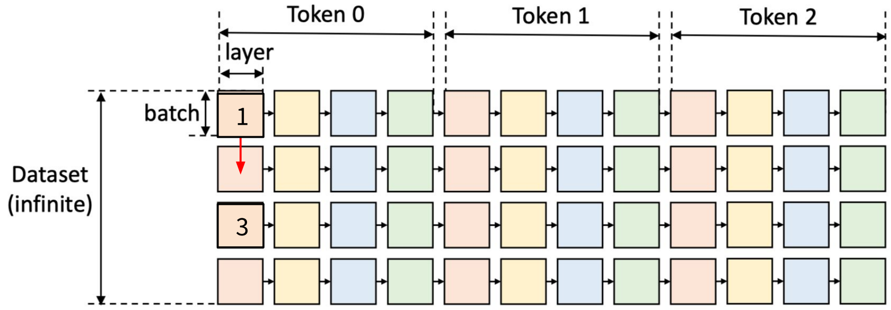
NVIDIA A100 40GB

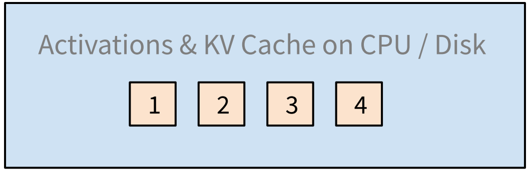
Throughput-Optimal Schedule

- Reuse layer weights
- Offload activations and KV cache



Overlapping Memory Access

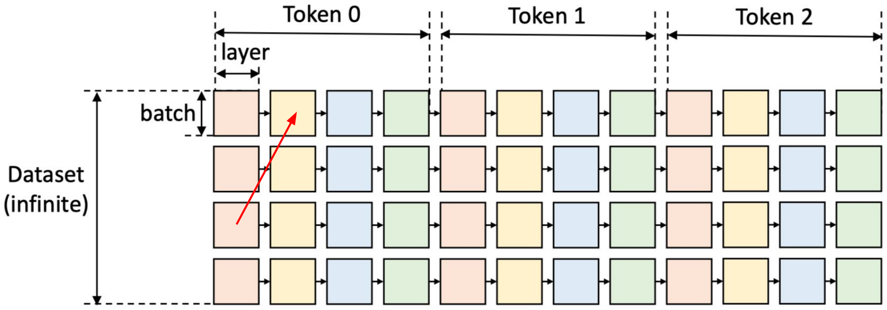




Algorithm 1 Block Schedule with Overlapping

for $i = 1$ **to** $generation\_length$ **do**
  for $j = 1$ **to** num_layers **do**
    // Compute a block with multiple GPU batches
    for $k = 1$ **to** $num\_GPU\_batches$ **do**
      // Load the weight of the next layer
      `load_weight`$(i, j + 1, k)$
      // Store the cache and activation of the prev batch
      `store_activation`$(i, j, k - 1)$
      `store_cache`$(i, j, k - 1)$
      // Load the cache and activation of the next batch
      `load_cache`$(i, j, k + 1)$
      `load_activation`$(i, j, k + 1)$
      // Compute this batch
      `compute`(i, j, k)
      // Synchronize all devices
      `synchronize`$()$
    end for
  end for
end for

Overlapping Memory Access



Algorithm 1 Block Schedule with Overlapping

```
for i = 1 to generation_length do
  for j = 1 to num_layers do
    // Compute a block with multiple GPU batches
    for k = 1 to num_GPU_batches do
      // Load the weight of the next layer
      load_weight(i, j + 1, k)
      // Store the cache and activation of the prev batch
      store_activation(i, j, k − 1)
      store_cache(i, j, k − 1)
      // Load the cache and activation of the next batch
      load_cache(i, j, k + 1)
      load_activation(i, j, k + 1)
      // Compute this batch
      compute(i, j, k)
      // Synchronize all devices
      synchronize()
    end for
  end for
end for
```

Tensor Placement

wg, wc, wd: percentage of **weights** placed on GPU, CPU, and disk

hg, hc, hd: **activations**

cg, cc, cd: **KV cache**

Tensor Placement

- Weights: layer granularity
- KV cache, activations: tensor granularity

Cost Model

$$T = T_{pre} \cdot l + T_{gen} \cdot (n - 1) \cdot l$$

T: latency

T_pre: prefilling latency for one layer (multiple blocks)

l: number of layers

T_gen: decoding latency for one layer (multiple blocks)

n: generation length

Cost Model

$$T_{pre} = \max(ctog^p, gtoc^p, dtoc^p, ctod^p, comp^p)$$

ctog: latency of transfer from CPU -> GPU, etc.

Cost Model

$$T_{pre} = \max(ctog^p, gtoc^p, dtoc^p, ctod^p, comp^p)$$

ctog: latency of transfer from CPU -> GPU, etc.

$$T_{gen} = \max(ctog^g, gtoc^g, dtoc^g, ctod^g, comp^g)$$

Cost Model Assumptions

- Perfect overlapping
- All latencies are estimated by summing up IO events
- Computation term is estimated by summing up matrix multiplication

Cost Model

Bls: effective batch size (batch size * # GPU blocks)

$$
\begin{aligned}
ctog^p &= \frac{weights\_ctog^p + act\_ctog^p}{ctog\_bdw} \\
&= \frac{1}{ctog\_bdw}\underline{((wc + wd)(8h_1^2 + 4h_1 \cdot h_2)} \\
&\quad \underline{+ 2(hc + hd)s \cdot h_1 \cdot bls)}
\end{aligned}
$$

Policy Search

$$
\begin{array}{lrcl}
\min\limits_{p} & & T/bls & \\
\text{s.t.} & gpu\ peak\ memory & < & gpu\ mem\ capacity \\
 & cpu\ peak\ memory & < & cpu\ mem\ capacity \\
 & disk\ peak\ memory & < & disk\ mem\ capacity \\
 & wg + wc + wd & = & 1 \\
 & cg + cc + cd & = & 1 \\
 & hg + hc + hd & = & 1
\end{array}
$$

CPU Compute

Size of KV cache: $b \times s \times h_1 \times 4$

Size of activation: $b \times h_1 \times 4$

IO reduction by s times if we compute attention scores on CPU

CPU Compute

$$f_{\text{Softmax}}\left(\frac{\mathbf{x}_Q^i \boxed{{\mathbf{x}_K^i}^T}}{\sqrt{h}}\right)\boxed{\cdot \mathbf{x}_V^i}$$

$$\boxed{\boxed{f_{\text{Softmax}}\left(\frac{\mathbf{x}_Q^i {\mathbf{x}_K^i}^T}{\sqrt{h}}\right)\cdot \mathbf{x}_V^i}}$$

Purple: CPU->GPU

Blue: CPU compute

Multiple GPUs

- Pipeline parallelism
  - Low communication costs
- Add micro-batches in inner loop

Algorithm 1 Block Schedule with Overlapping

for $i = 1$ **to** $generation\_length$ **do**
  for $j = 1$ **to** num_layers **do**
    // Compute a block with multiple GPU batches
    for $k = 1$ **to** $num\_GPU\_batches$ **do**
      // Load the weight of the next layer
      `load_weight`$(i, j + 1, k)$
      // Store the cache and activation of the prev batch
      `store_activation`$(i, j, k - 1)$
      `store_cache`$(i, j, k - 1)$
      // Load the cache and activation of the next batch
      `load_cache`$(i, j, k + 1)$
      `load_activation`$(i, j, k + 1)$
      // Compute this batch
      `compute`(i, j, k)
      // Synchronize all devices
      `synchronize`$()$
    end for
  end for
end for

Quantization

- Group quantization to 4 bits for weights and KV cache

$$x_{quant} = round\left(\frac{x - min}{max - min} \times (2^b - 1)\right)$$

- Group weights along output dim and KV cache along hidden dim
- CPU compute is turned off due to overhead
- Further reduces IO cost

Sparse Attention

- Compute scores given Q
- Only load top 10% K with highest scores
- Select corresponding Vs

Experimental Setup

- T4 GCP instances
- 2 GB/s read and 1 GB/s write for SSD
- OPT models: 6.7B, 30B, 175B
- Synthetic dataset with padded prompts
- Prompt length: 512, 1024
- Generation length: 32

Device	Model	Memory
GPU	NVIDIA T4	16 GB
CPU	Intel Xeon @ 2.00GHz	208 GB
Disk	Cloud default SSD (NVMe)	1.5 TB

Baseline

- DeepSpeed Inference
- HuggingFace Accelerate
- Petals
  - Collaborative inference over network
  - 10 ms latency and 1Gbps bandwidth

Throughput Results

- Petals uses 1, 4, and 24 GPUs respectively for the three model sizes

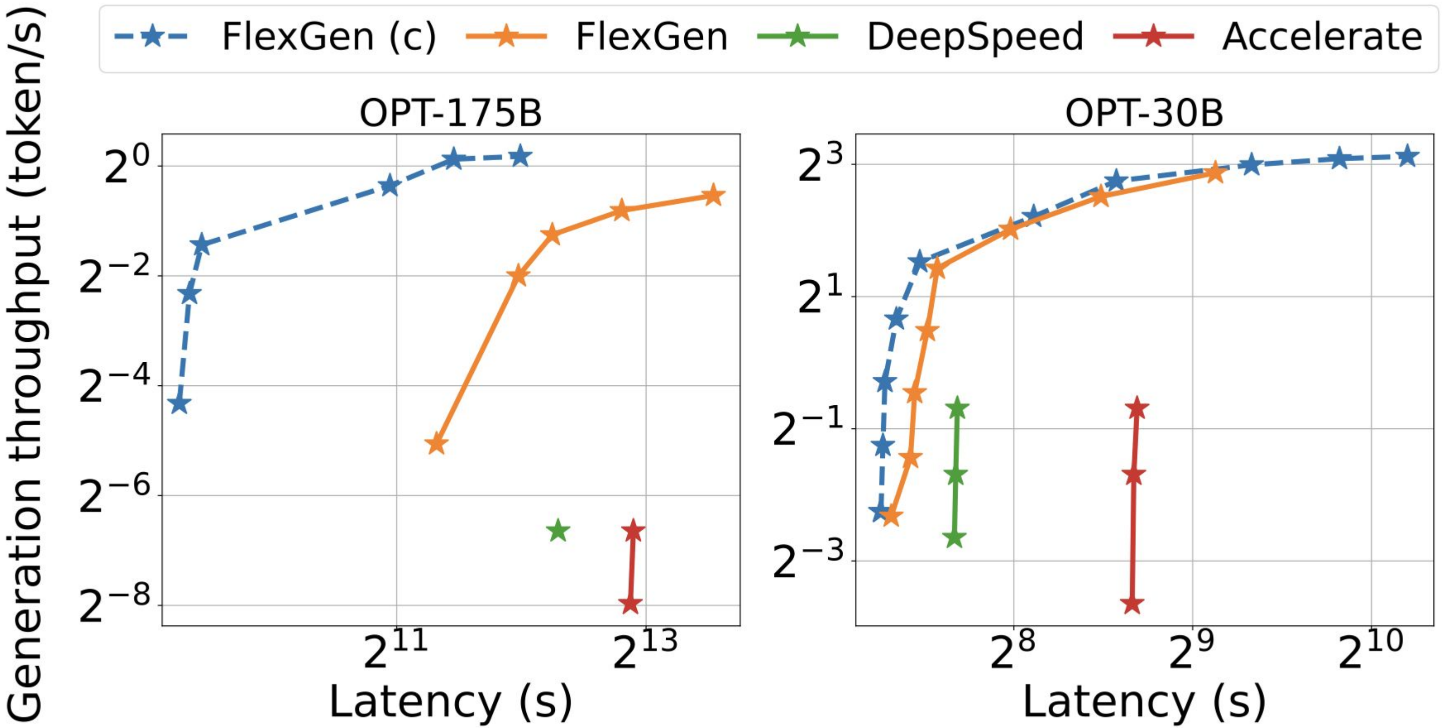
Seq. length	512			1024		
Model size	6.7B	30B	175B	6.7B	30B	175B
Accelerate	25.12	0.62	0.01	13.01	0.31	0.01
DeepSpeed	9.28	0.60	0.01	4.59	0.29	OOM
Petals	8.25	2.84	0.08	6.56	1.51	0.06
FlexGen	25.26	7.32	0.69	13.72	3.50	0.35
FlexGen (c)	29.12	8.70	1.12	13.18	3.98	0.42

Multi-GPU Scaling

- 4 GPUs, per GPU throughput reported
- Superlinear scaling for decoding throughput with pipeline parallelism
- Generation (prefilling + decode) doesn't scale superlinearly with n = 32

Metric	Generation Throughput			Decoding Throughput		
Model size	6.7B	30B	175B	6.7B	30B	175B
FlexGen (1)	25.26	7.32	0.69	38.28	11.52	0.83
FlexGen (4)	201.12	23.61	2.33	764.65	48.94	3.86
DeepSpeed (4)	50.00	6.40	0.05	50.20	6.40	0.05

Latency-Throughput Tradeoff

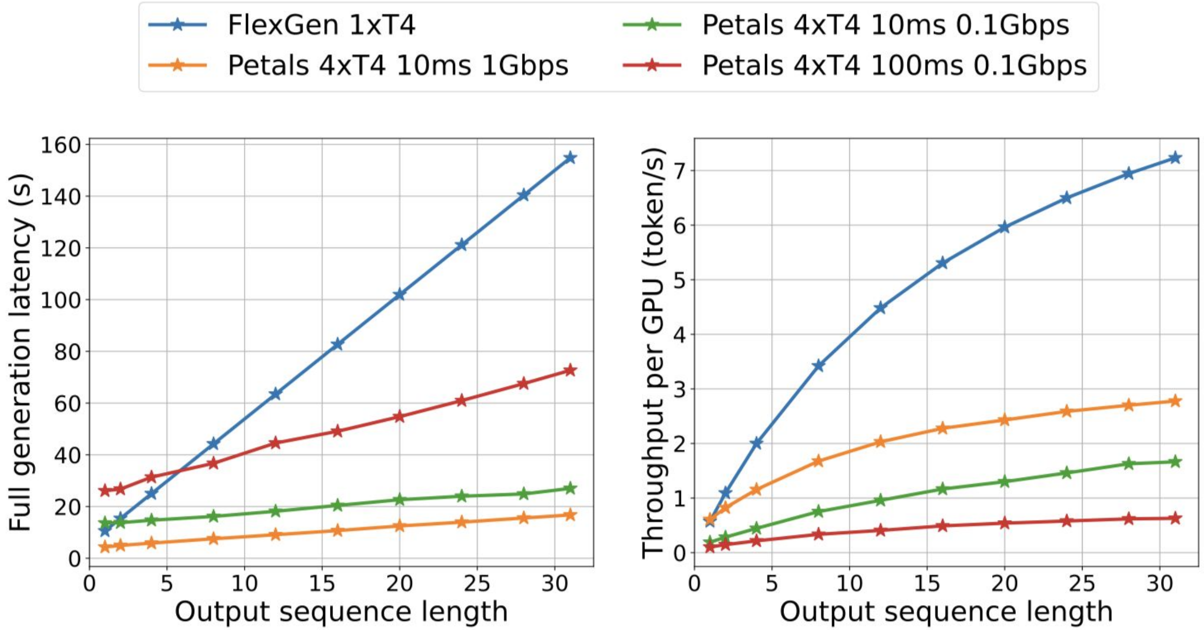


Quantization Results

Table 5. The accuracy (higher is better) and perplexity (lower is better) with approximate methods.

Dataset	Lambada (acc)			WikiText (ppl)		
Config	FP16	4-bit	4-bit-S	FP16	4-bit	4-bit-S
OPT-30B	0.725	0.724	0.718	12.72	12.90	12.90
OPT-175B	0.758	0.756	0.756	10.82	10.94	10.94

Comparisons with Collaborative Inference



Ablation Study

Table 23. Ablation study of proposed techniques. The numbers are generation throughput on 1 T4 GPU with prompt length 512 and generating length 32. The gray tuple denotes a policy (GPU batch size × #GPU-batch, wg, wc, cg, cc, hg, hc).

Model size	30B	175B
All optimizations	7.32 (48×3, 20, 80, 0, 100, 0, 100)	0.69 (32×8, 0, 50, 0, 0, 0, 100)
No policy search	$\boxed{7.26}$ (48×3, 0, 100, 0, 100, 0, 100)	$\boxed{0.27}$ (32×1, 0, 50, 0, 0, 0, 100)
No overlapping	5.86 (48×3, 20, 80, 0, 100, 0, 100)	0.59 (32×8, 0, 50, 0, 0, 0, 100)
No CPU compute	4.03 (48×3, 20, 80, 0, 100, 0, 100)	0.62 (32×8, 0, 50, 0, 0, 0, 100)
No disk	7.32 (48×3, 20, 80, 0, 100, 0, 100)	OOM
w/ DeepSpeed policy	1.57 (8×1, 0, 100, 100, 0, 100, 0)	0.01 (2×1, 0, 0, 100, 0, 100, 0)

Runtime Breakdown

Table 8. Execution time breakdown (seconds) for OPT-175B. The prompt length is 512. (R) denotes read and (W) denotes write.

Stage	Total	Compute	Weight (R)	Cache (R)	Cache (W)
Prefill	2711	2220	768	0	261
Decoding	11315	1498	3047	7046	124

Compression Throughput

(batch x bls, wg, wc, cg, cc, hg, hc)

Seq. length	512 + 32		
Model size	6.7B	30B	175B
Accelerate	183.177 (16×1, 100, 0, 100, 0, 100, 0)	2.077 (13×1, 0, 100, 100, 0, 100, 0)	0.026 (4×1, 0, 0, 100, 0, 100, 0)
DeepSpeed	38.027 (32×1, 0, 100, 100, 0, 100, 0)	3.889 (12×1, 0, 100, 100, 0, 100, 0)	0.019 (3×1, 0, 0, 100, 0, 100, 0)
FlexGen	233.756 (28×1, 100, 0, 100, 0, 100, 0)	5.726 (4×15, 25, 75, 40, 60, 100, 0)	0.384 (64×4, 0, 25, 0, 0, 100, 0)
FlexGen (c)	$\boxed{120.178}$ (144×1, 100, 0, 100, 0, 100, 0)	16.547 (96×2, 25, 75, 0, 100, 100, 0)	1.114 (24×1, 0, 100, 0, 100, 100, 0)

Takeaways

- Efficient offload strategy
  - Formulate cost model and search space for offloading policy
- 4-bit quantization on KV cache and weights
- Scaled to multi-GPU with pipeline parallelism