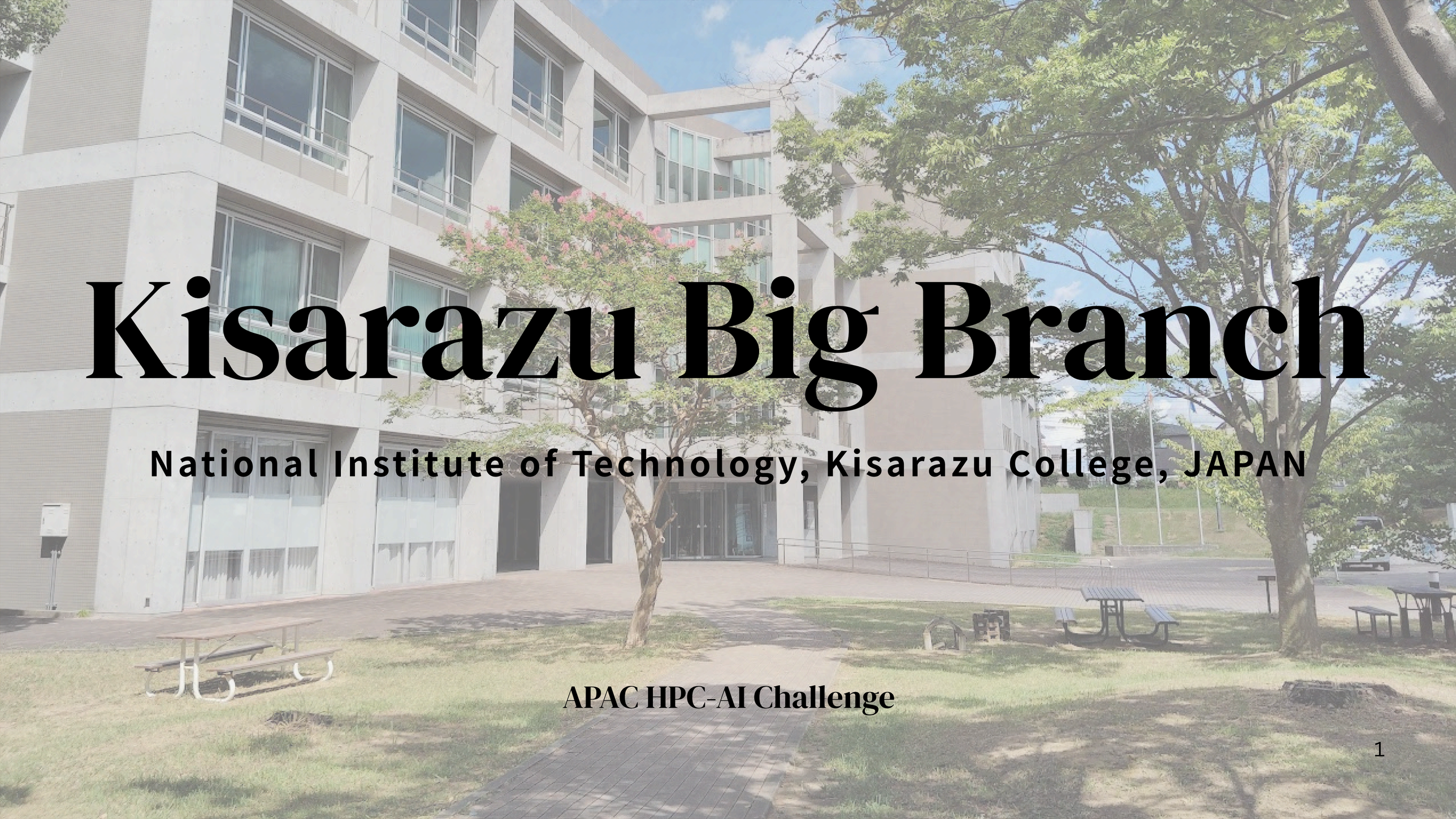


Kisarazu Big Branch

National Institute of Technology, Kisarazu College, JAPAN

APAC HPC-AI Challenge

Table of Contents

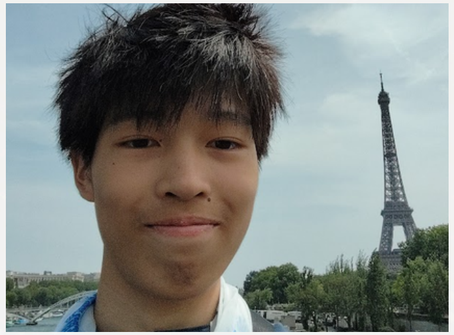
Table of Contents

Our Team

Kisarazu Big Branch

Ochi Yuma (Leader)

I’m eating Kaya Toast

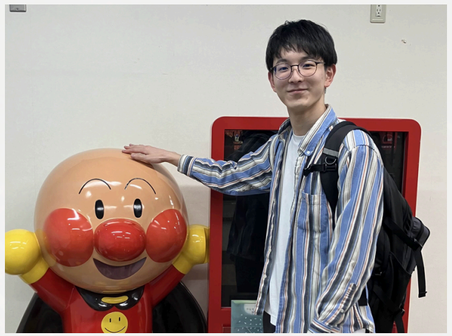


Tsuneki Hirotaka

Computer OTAKU

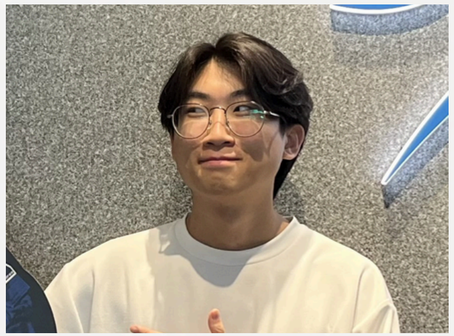
Akimoto Sora

Goodnight



Naito Masahiro

I like Anpanman



Isaac Yap Zhen Khai

That one foreigner on the team

Prof. Oeda Shinichi

Our Glorious Supreme Big Branch Supervisor

Objective: Maximizing throughput of DeepSeek 671B inference

Framework:

SGLang

Machine:

Aspire2a+

- 8x H100 / node
- 2 nodes access

Constraints:

No experimental features
(eg: torch.compile)

SGLang

A fast serving framework for large language models and vision language models

Features

- Fast Backend Runtime
- Flexible Frontend Language
- Extensive Model Support
- Active Community

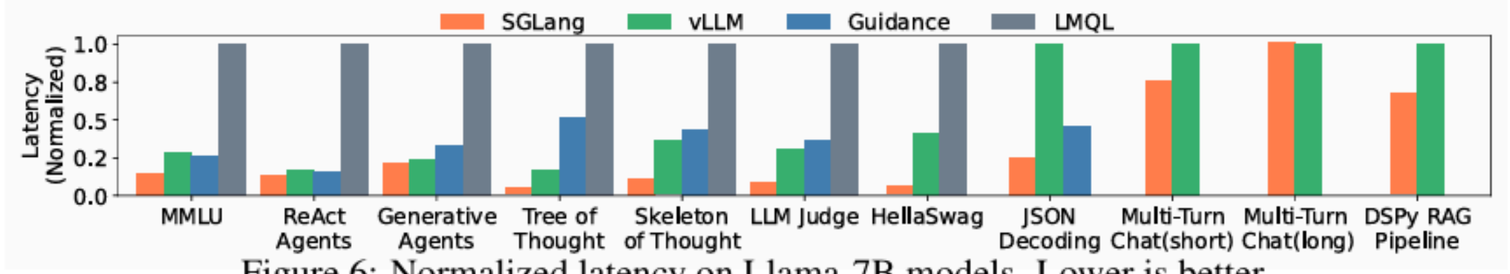


Figure 6: Normalized latency on Llama-7B models. Lower is better.

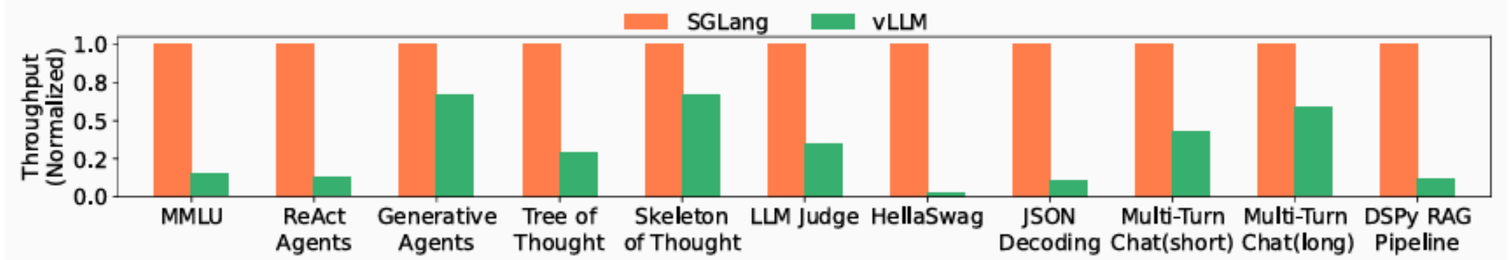


Figure 7: Normalized throughput on Mixtral-8x7B models with tensor parallelism. Higher is better.

https://arxiv.org/pdf/2312.07104

Table of Contents

DeepSeek R1

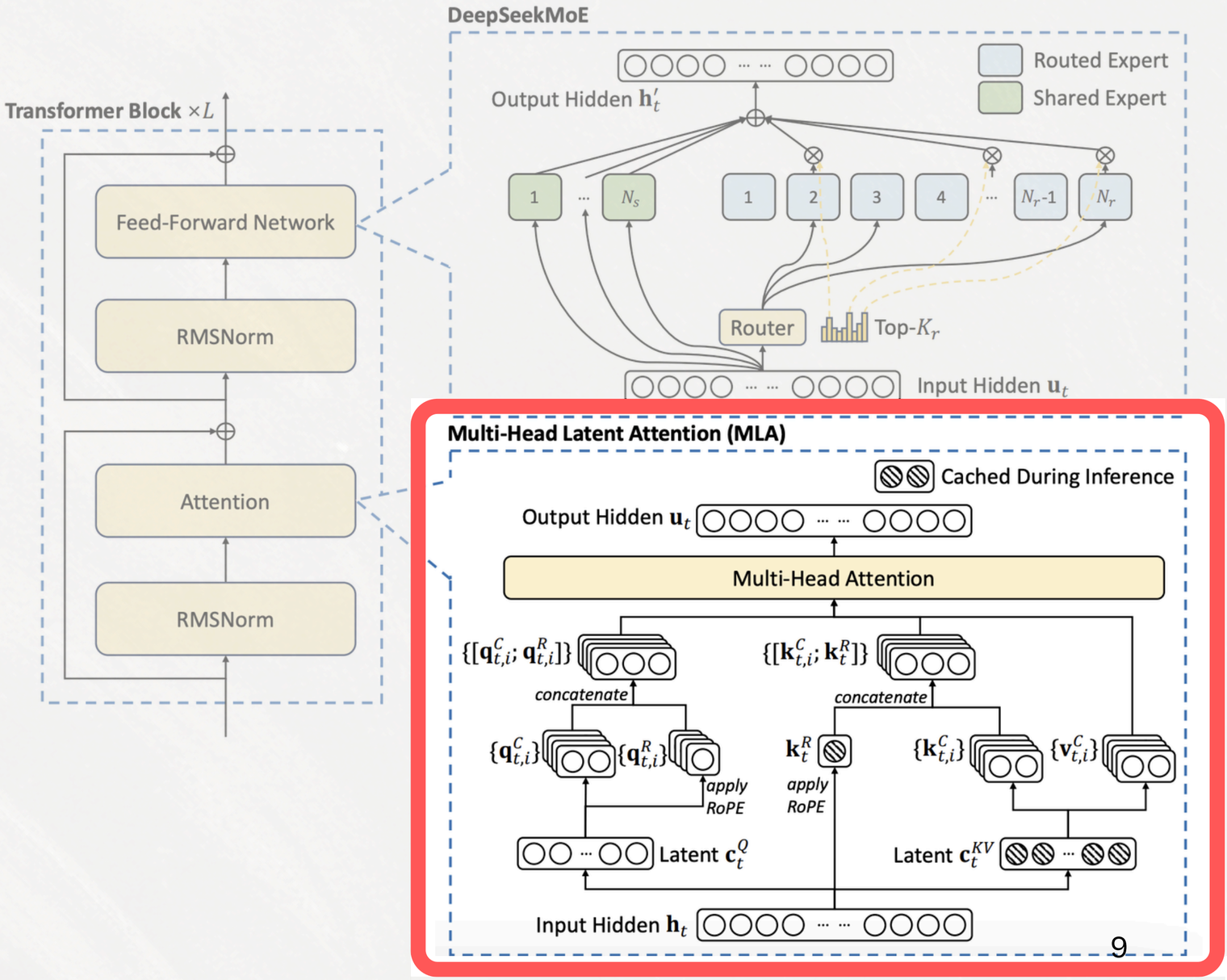
1. 671B parameters,
   37B active

2. Applied Reinforcement Learning during Training

3. Key features include
   - Multi-head Latent Attention (MLA)
   - Mixture of Experts (MoE)
   - Multi-Token Prediction (MTP)

Multi-head Latent Attention (MLA)

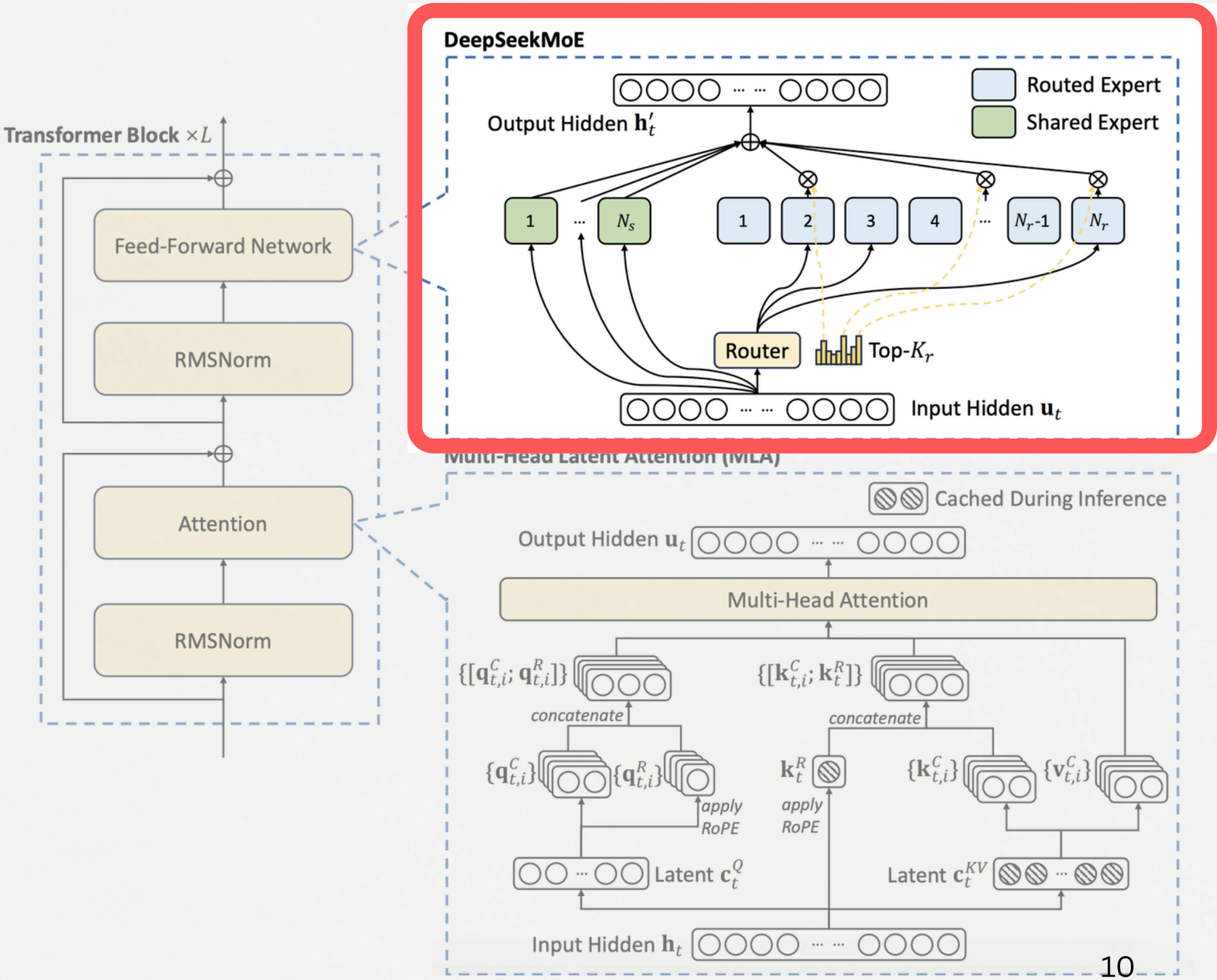
- Compresses KV cache into lower-dimensional latent space
- Boosts training and inference speed while significantly reducing memory usage



https://arxiv.org/pdf/2412.19437

Mixture of Experts (MoE)

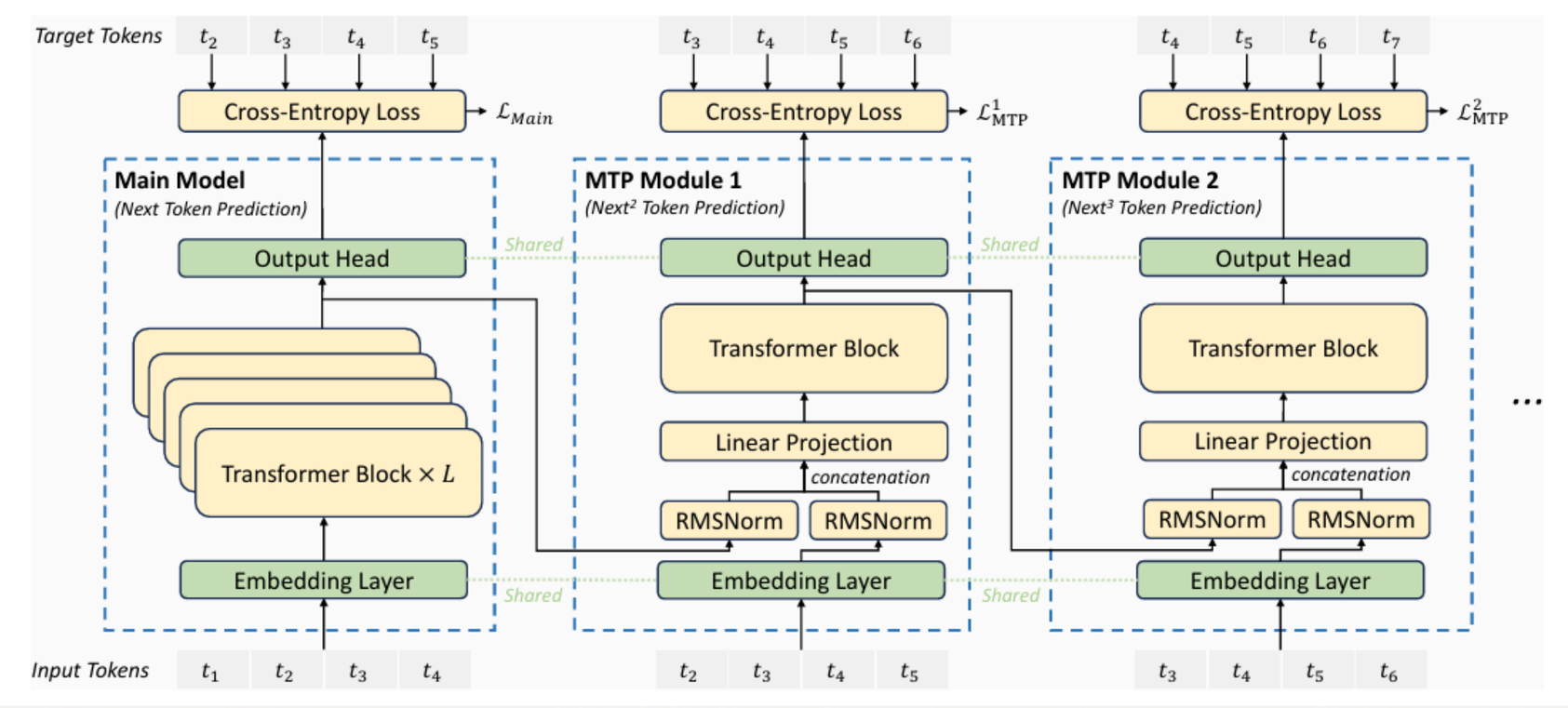
- Replaces FFN with certain number of NN (“experts”)
- Router determines which tokens are sent to which experts
- Efficient pretraining and faster inference owing to only partial usage of parameters



https://arxiv.org/pdf/2412.19437

Multi-Token Prediction (MTP)

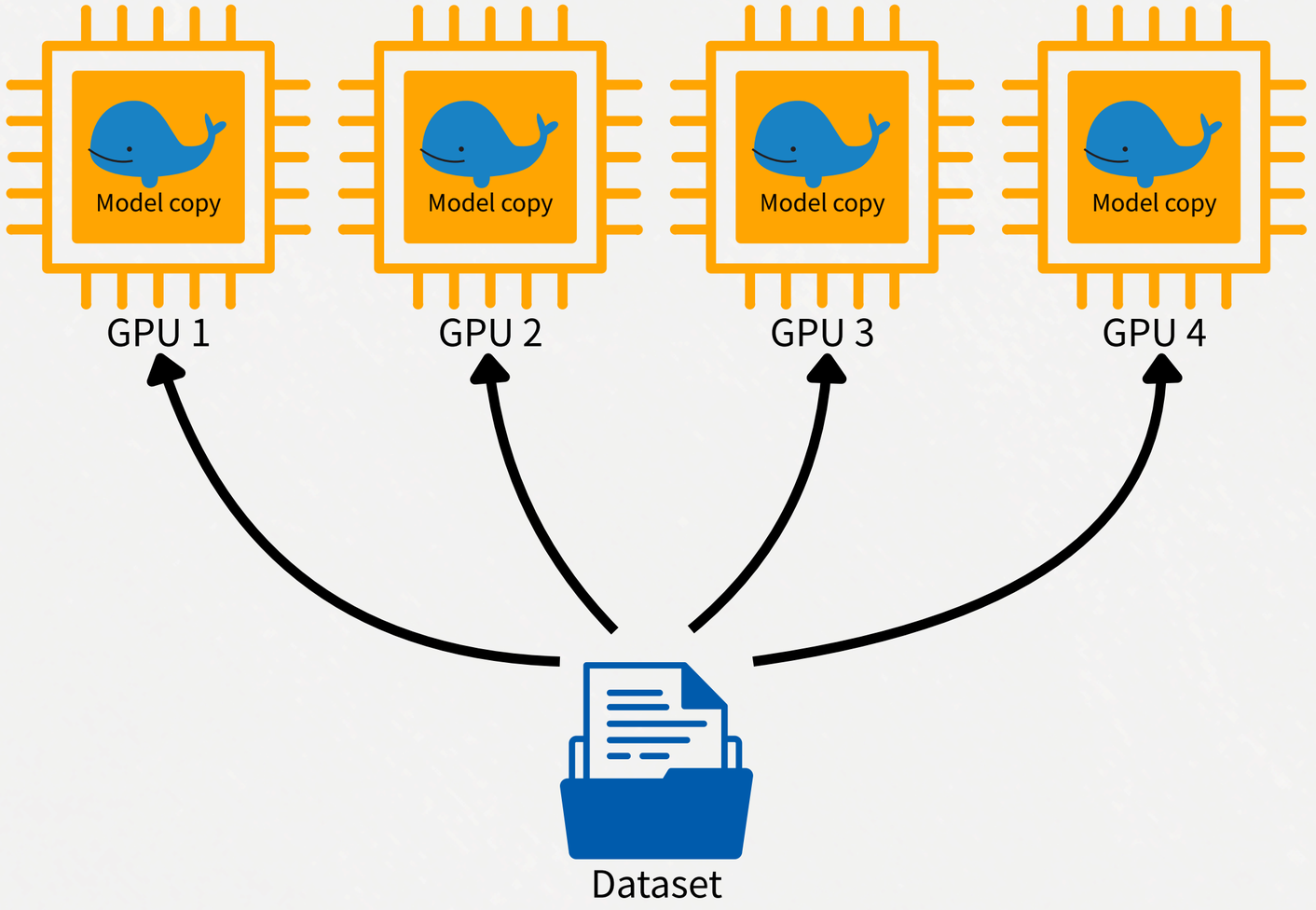
- Extends the prediction scope to multiple future tokens at each position.
- Allows speculative decoding to improve inference speed by predicting multiple tokens at once



https://arxiv.org/pdf/2412.19437

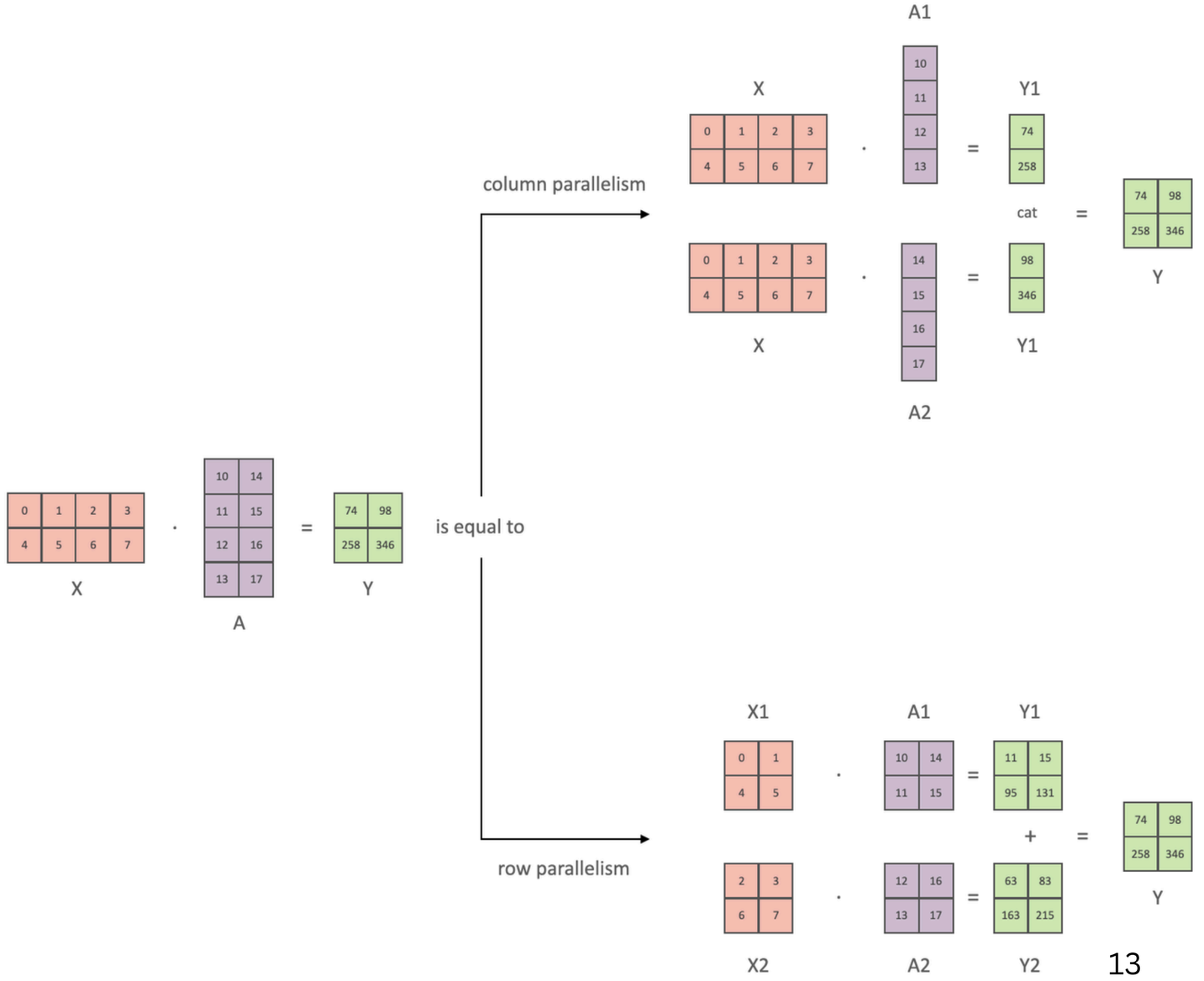
Data Parallelism (DP)

- Replicates model across GPUs
- Data batches are evenly distributed
- Computation workload is efficiently shared, leading to faster inference



Tensor Parallelism (TP)

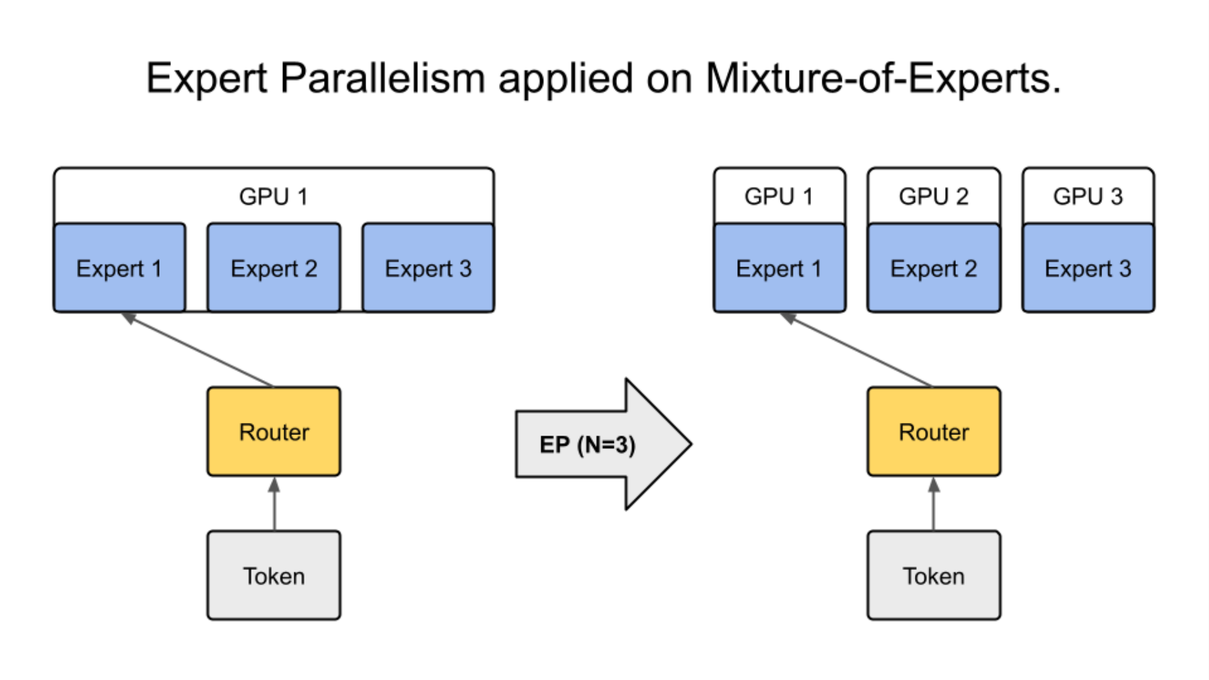
- Shards the model’s tensors across GPUs
- Enables large models that are not able to fit in one GPU to fit on multiple GPUs
- Increases overhead as inter-GPU communication is required



https://huggingface.co/docs/transformers/v4.15.0/parallelism

Expert Parallelism (EP)

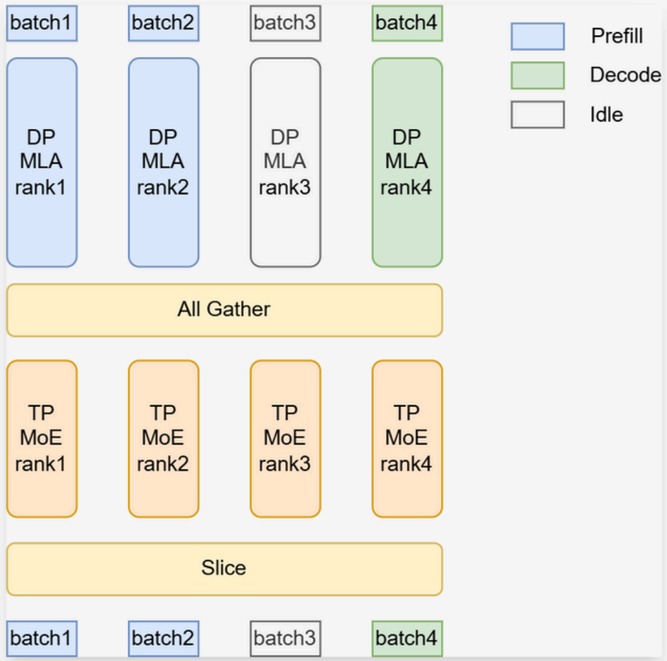
- Distributes whole MoE experts (that are not affected by TP) across GPUs
- Only applies to MoE layers and does not affect other layers



https://docs.nvidia.com/nemo-framework/user-guide/latest/nemotoolkit/features/parallelisms.html

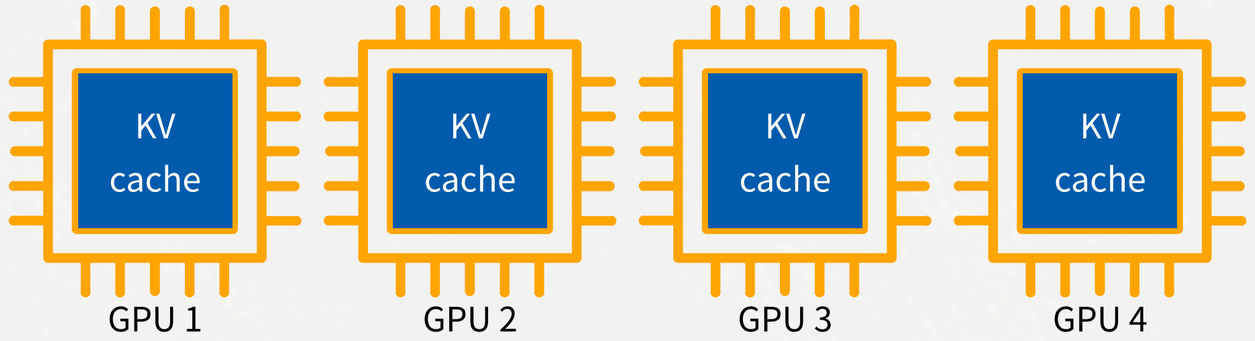
DP Attention

- DPs the attention across GPUs
- Splits the KV cache across GPUs, reducing memory usage

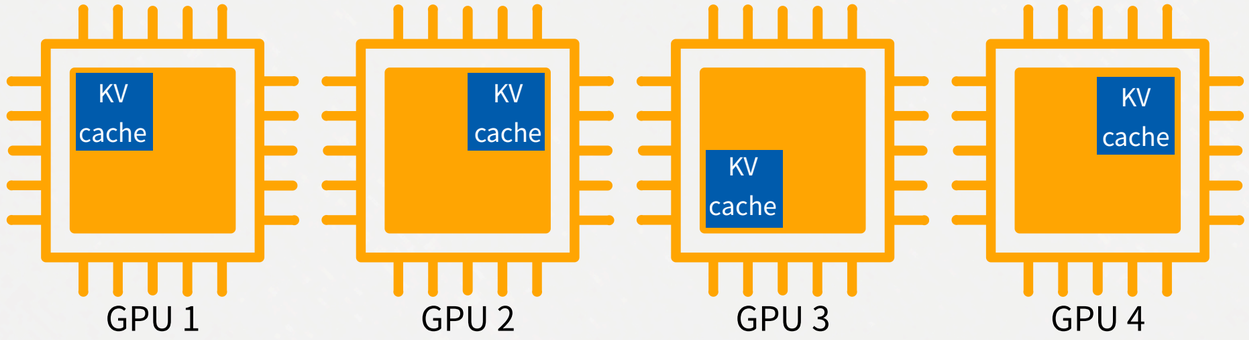


https://lmsys.org/blog/2024-12-04-sglang-v0-4/

Normal TP



DP Attention



- KV cache memory usage is reduced by sharding it across GPUs

Table of Contents

Our Baseline -Base 0-

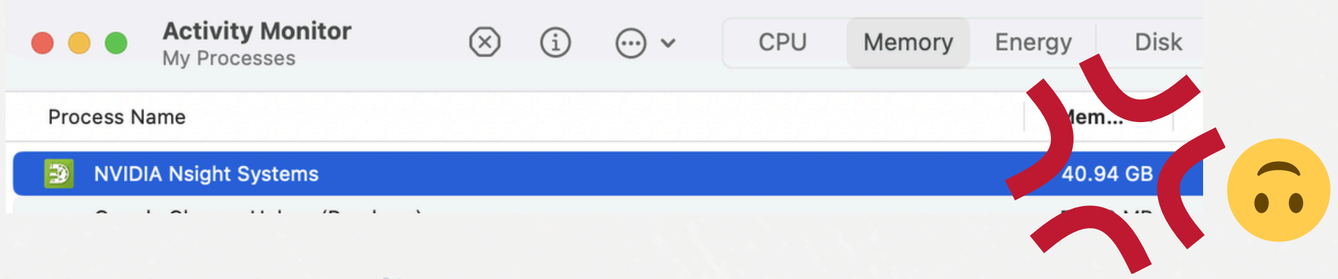
with example jobscript specified in task rules **(SGLang v0.4.10 used)**

- 6347.38 (tok/s)

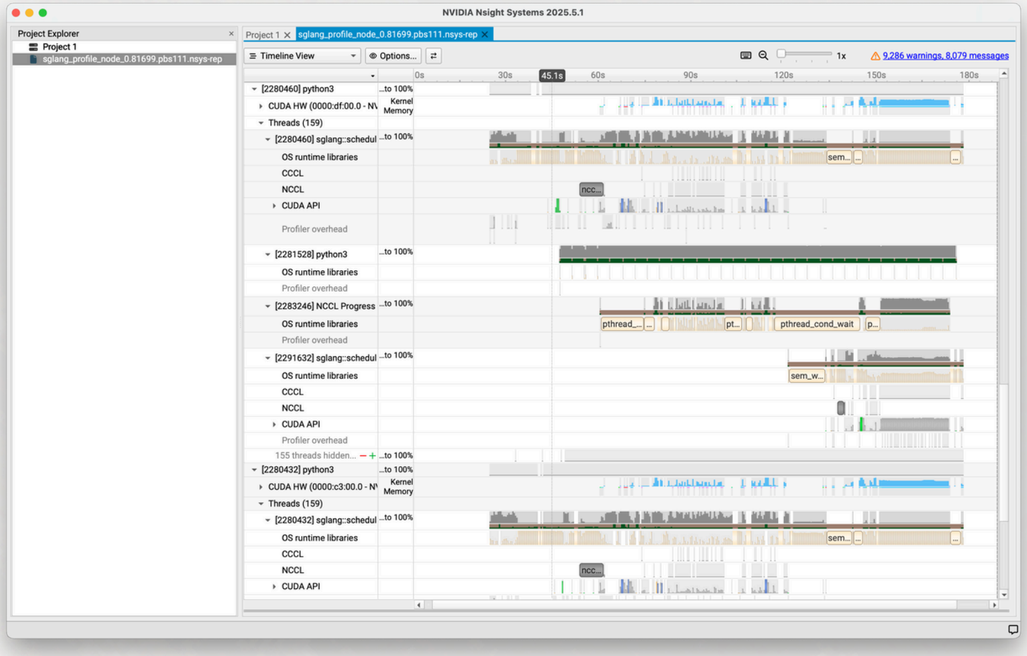
```
#!/bin/bash
#PBS -P 50000130
#PBS -l walltime=00:10:00
#PBS -l select=2:ncpus=112:ngpus=8:mpiprocs=2:mem=1880gb
#PBS -j oe
#PBS -m abe
#PBS -o output.txt
##PBS -l other=hyperthread

module load cuda/12.2.2 cudnn/12-9.3.0.75 nccl/12.6-2.22.3-1

NSYS_PATH="/home/users/industry/ai-hpc/apacsc11/scratch/nsys/target-linux-x64/nsys"

cd ${HOME}/ochi/solve

/usr/mpi/gcc/openmpi-4.1.7a1/bin/mpirun \
-hostfile ${PBS_NODEFILE} \
-map-by ppr:1:node:PE=112 -oversubscribe -use-hwthread-cpus \
-bind-to none --report-bindings -display-map \
-tag-output -output-filename ${HOME}/run/sglang.${PBS_JOBID} \
-x PATH \
-x NCCL_DEBUG=INFO \
bash -c "uv run python \
-m sglang.bench_offline_throughput \
--model-path /home/users/industry/ai-hpc/apacsc11/scratch/deepseek \
--dataset-path /home/users/industry/ai-hpc/apacsc11/scratch/sharegpt/ShareGPT_V3_unfiltered_cleaned_split.json \
--num-prompts 2000 --load-format dummy --seed 2025 --dtype bfloat16 \
--tp 16 --nnodes 2 --trust-remote-code \
--dist-init-addr \${DIST_INIT_ADDR}:5000 --node-rank \${OMPI_COMM_WORLD_RANK}"
```

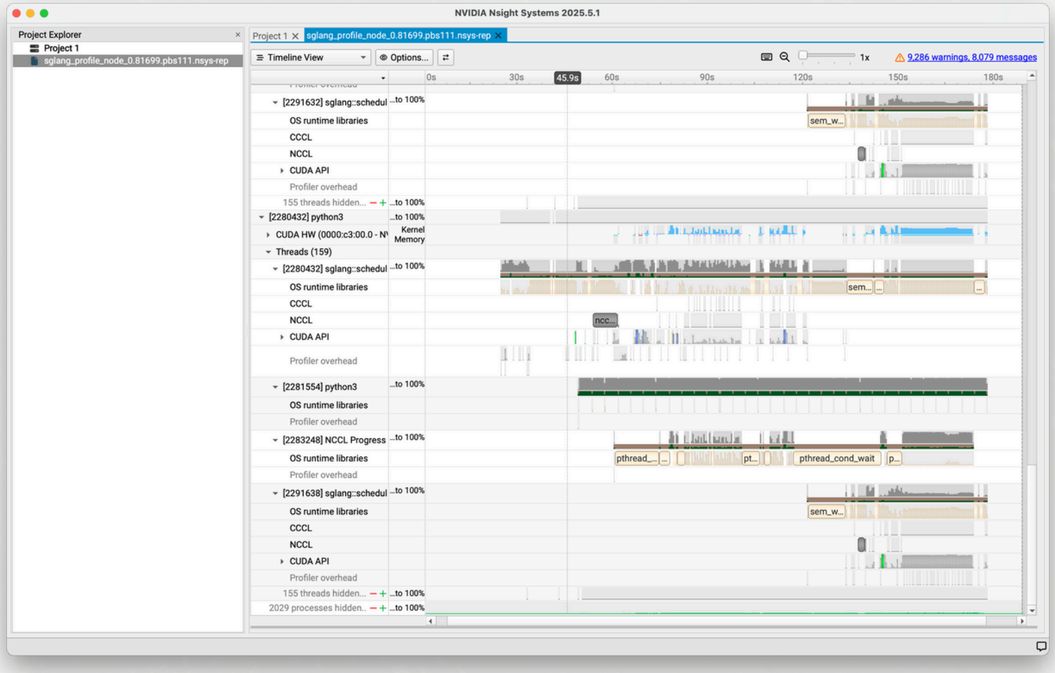
```
[1,0]<stdout>:#Input tokens: 626729
[1,0]<stdout>:#Output tokens: 388685
[1,0]<stdout>:#Input tokens: 4096
[1,0]<stdout>:#Output tokens: 256
[1,0]<stdout>:
[1,0]<stdout>:====== Offline Throughput Benchmark Result =======
[1,0]<stdout>:Backend:                                 engine
[1,0]<stdout>:Successful requests:                     2000
[1,0]<stdout>:Benchmark duration (s):                  159.97
[1,0]<stdout>:Total input tokens:                      626729
[1,0]<stdout>:Total generated tokens:                  388685
[1,0]<stdout>:Last generation throughput (tok/s):      21.19
[1,0]<stdout>:Request throughput (req/s):              12.50
[1,0]<stdout>:Input token throughput (tok/s):          3917.70
[1,0]<stdout>:Output token throughput (tok/s):         2429.68
[1,0]<stdout>:Total token throughput (tok/s):          6347.38
[1,0]<stdout>:-------------------------------------------------
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                        Resource Usage on 2025-09-04 23:05:54.180
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        JobId: 83373.pbs111
        Project: 50000130
        Exit Status: 0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        NCPUs: Requested(224), Used(224)
        CPU Time Used: 00:49:42
        Memory: Requested(3760gb), Used(37195476kb)
        Vmem Used: 19948242224kb
        Walltime: Requested(00:10:00), Used(00:06:05)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Profiling Results

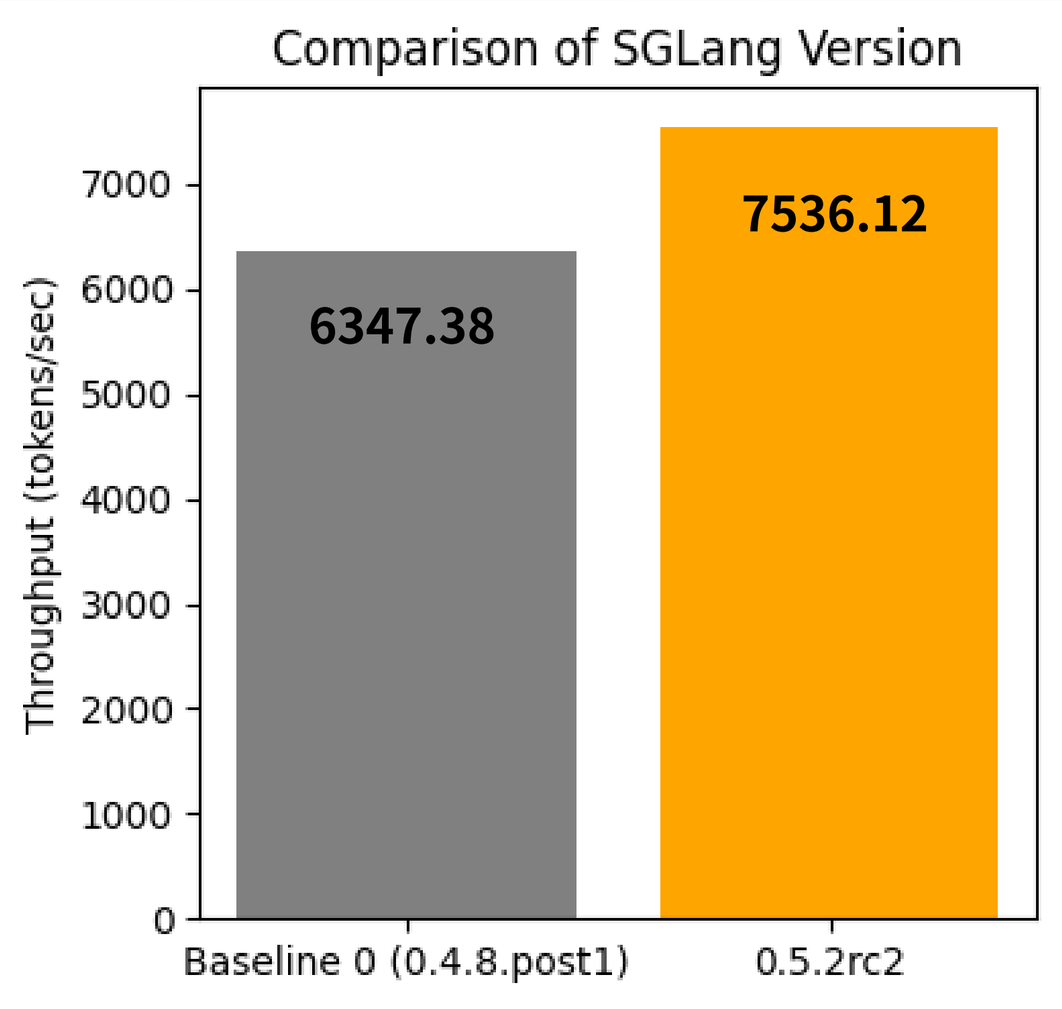


- Why profiling doesn't make sense for us
  - Too many factors, we couldn't identify where the bottlenecks are🤯.
  - Nsight Systems consumes too much memory (e.g., 40GB consumption for ONLY 200 requests profiling data, required is 2000)





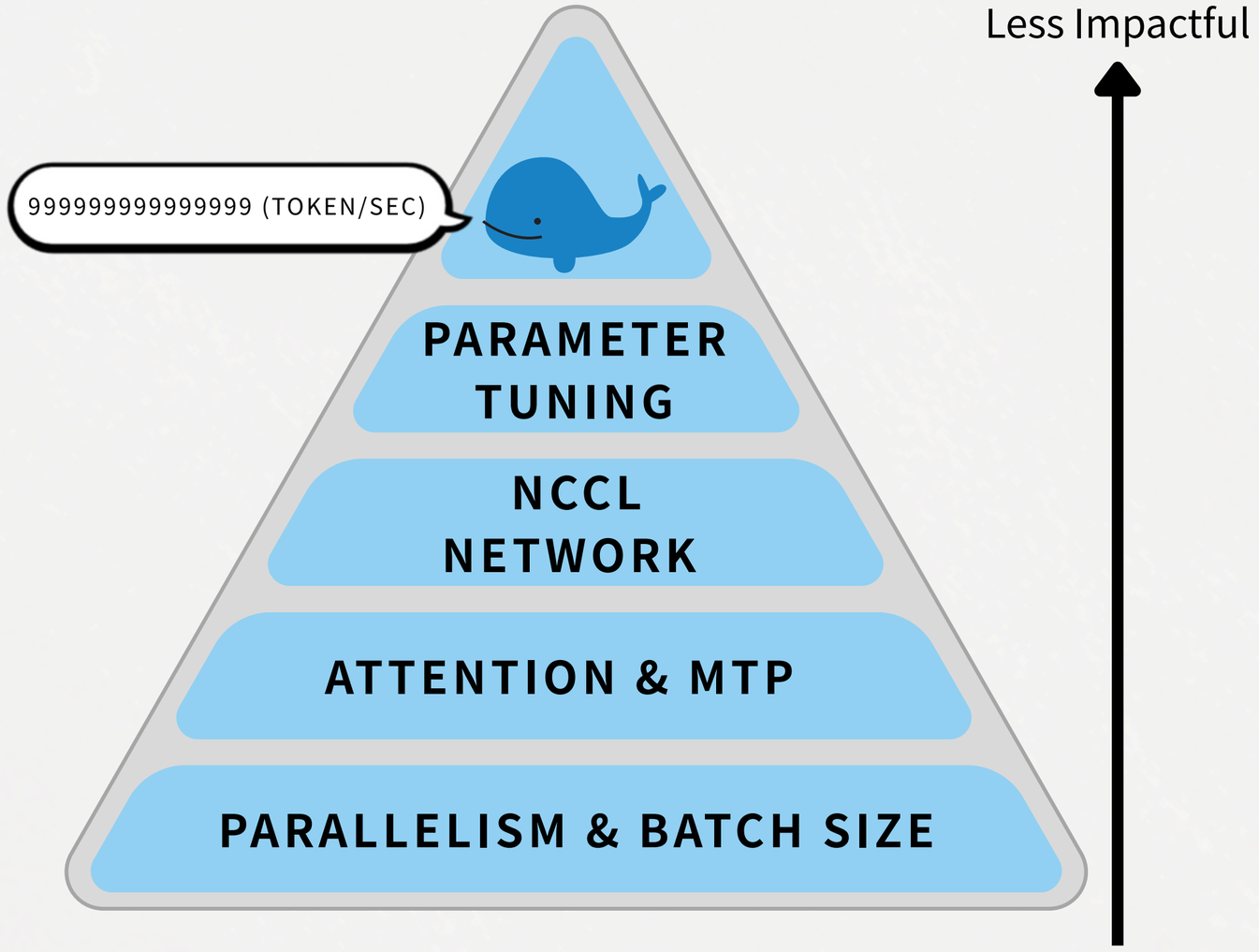
Updated SGLang → -Base 1-

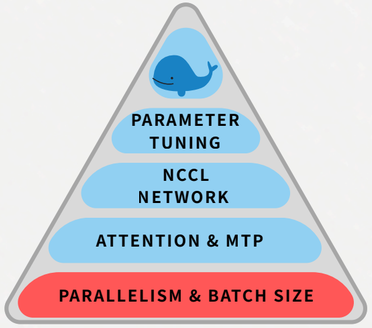


- Accelerating SGLang with Multiple Token Prediction (v0.4.10)
- SGLang HiCache: Fast Hierarchical KV Caching with Your Favorite Storage Backends (v0.5.2)

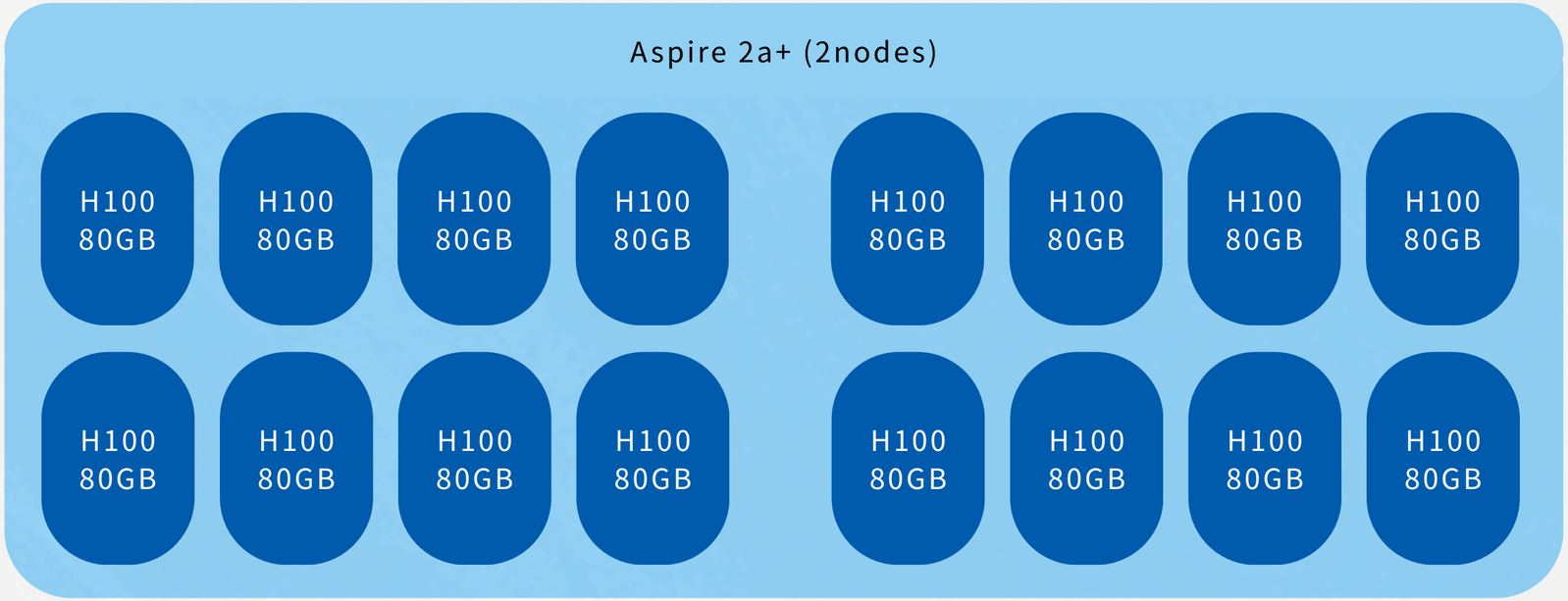
Inference Optimization Strategy

- Hierarchically classified optimization method by impactfulness
- In the absence of profiling, we scour logs to look for various pieces of information, such as GPU memory usage, number of cached tokens, etc.
- Our experiments and next steps are decided based heavily on our findings from logs





DP & TP Configuration

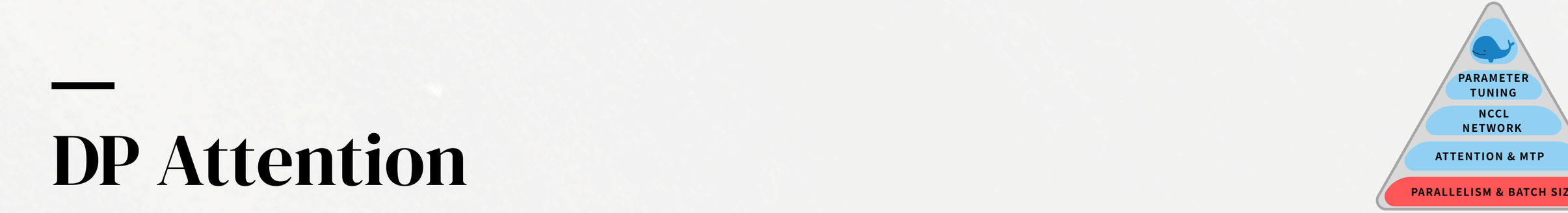


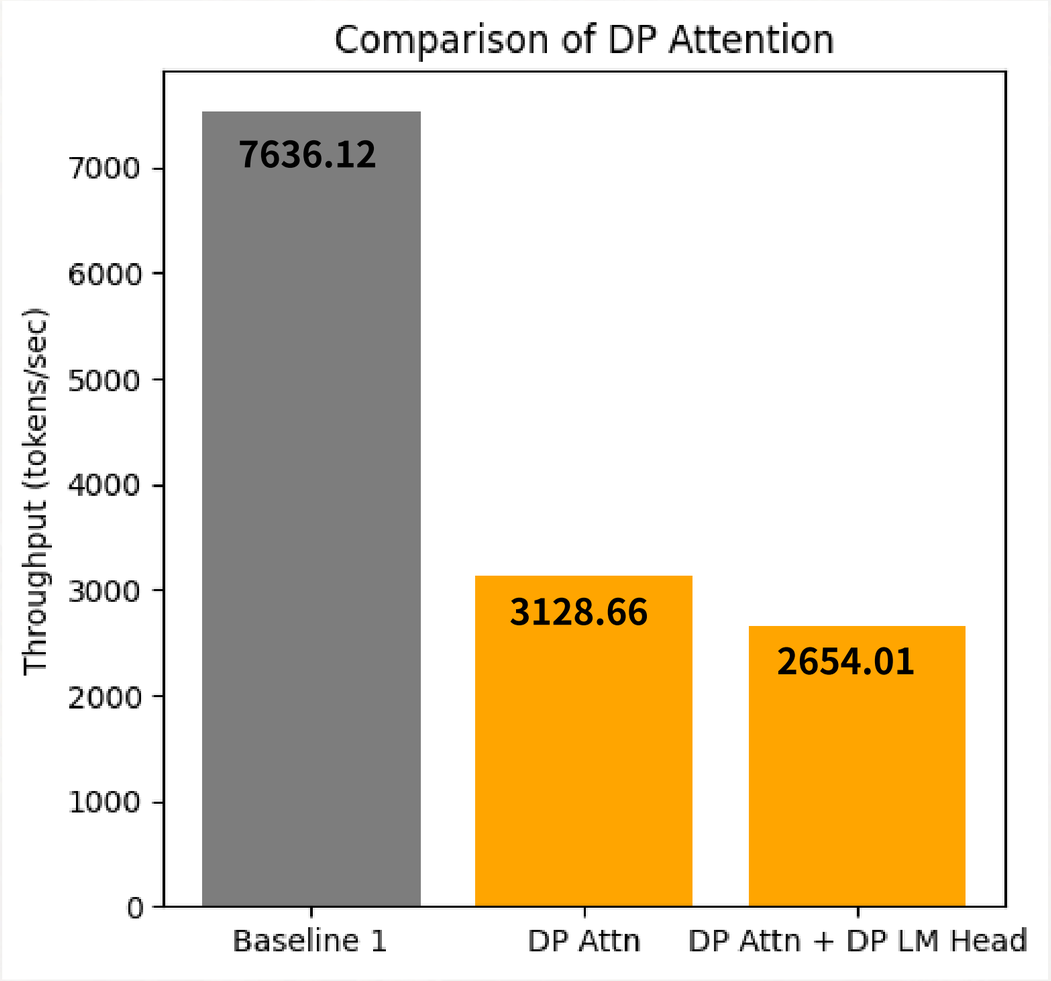
1,280GB (80GB x 16)

Up to 1,342GB

Only DP=1, TP=16 was possible

DP Attention





Our hypothesis:

- Increased inter-GPU communication overhead due to increased all-gather operations

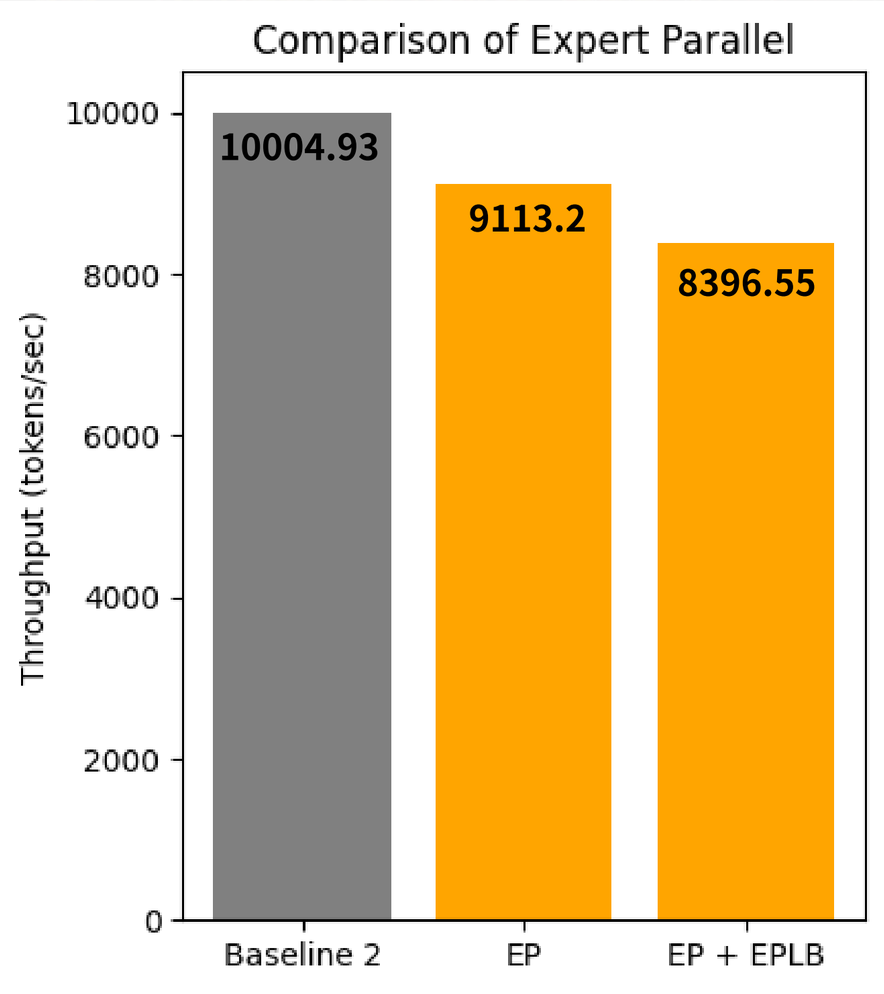
```
13 TP12] KV Cache is allocated. #tokens: 424274, KV size: 27.77 GB
13 TP1] KV Cache is allocated. #tokens: 424274, KV size: 27.77 GB
13 TP8] KV Cache is allocated. #tokens: 424274, KV size: 27.77 GB
13 TP10] KV Cache is allocated. #tokens: 424274, KV size: 27.77 GB
13 TP13] KV Cache is allocated. #tokens: 424274, KV size: 27.77 GB
```

Before DP Attention

```
TP13] KV Cache is allocated. #tokens: 214770, KV size: 14.06 GB
TP9] KV Cache is allocated. #tokens: 214770, KV size: 14.06 GB
TP15] KV Cache is allocated. #tokens: 214770, KV size: 14.06 GB
TP14] KV Cache is allocated. #tokens: 214770, KV size: 14.06 GB
TP12] KV Cache is allocated. #tokens: 214770, KV size: 14.06 GB
```

After DP Attention

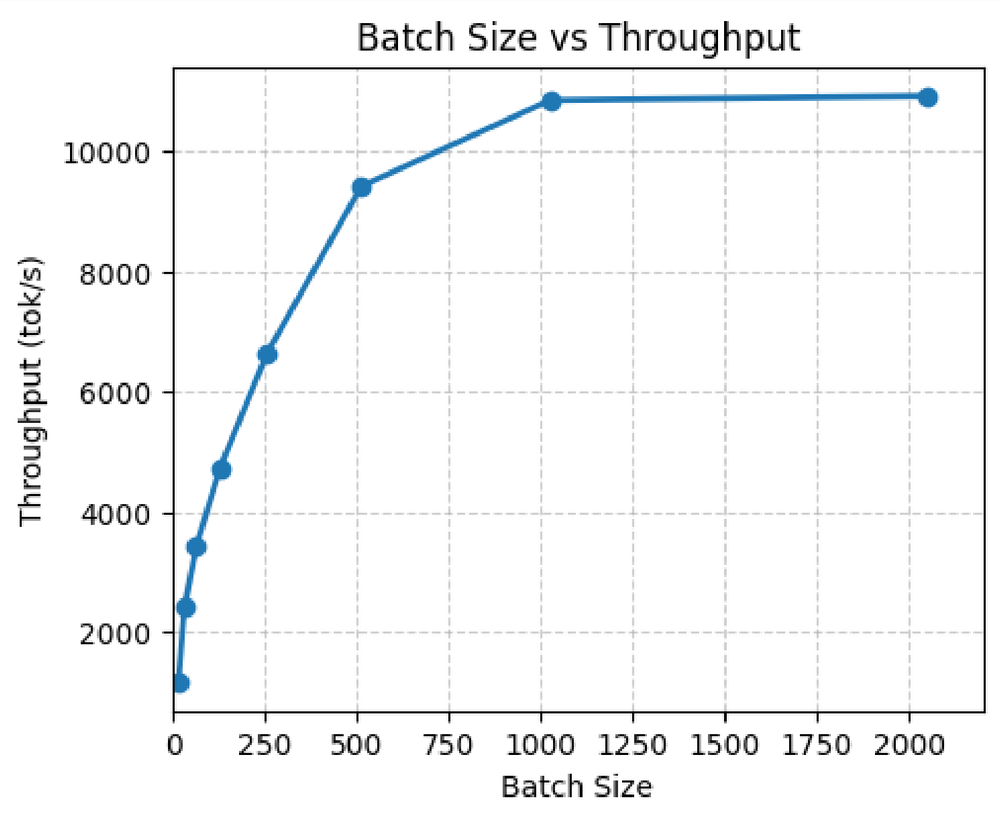
Expert Parallelism



Our hypothesis:

- Overhead introduced by EP and EPLB is more than the actual speedup
- EP may lead to GPU bottlenecks if only certain experts are preferred
- EPLB helps to curb this issue but itself introduces much overhead

Batch Size



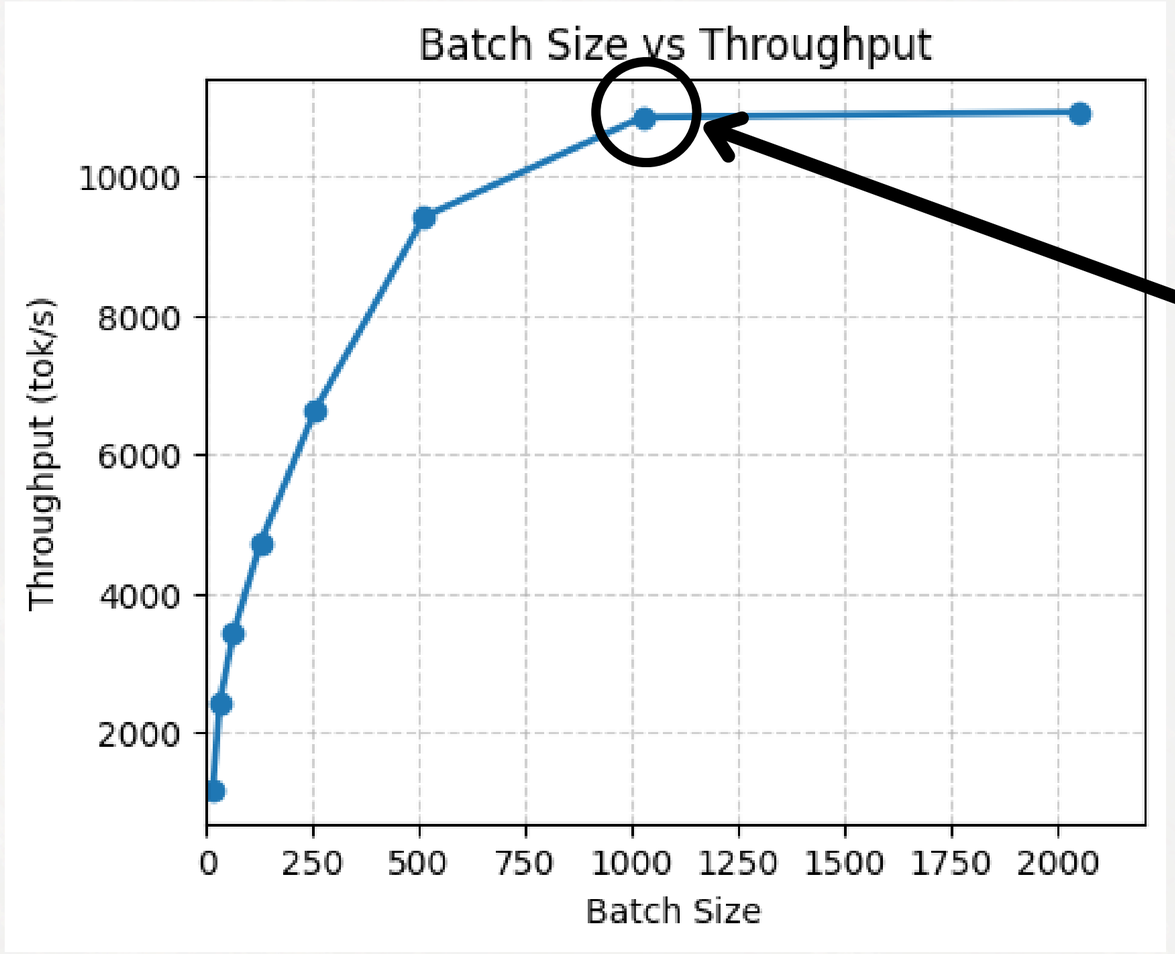
We observed that throughput increases with batch size

```
other_args.extend(
    [
        "--cuda-graph-max-bs",
        batch_size,
        "--mem-fraction-static",
        server_args.mem_fraction_static,
        "--tp-size",
        server_args.tp_size,
        "--max-running-requests",
        batch_size,
    ]
)
```

In SGLang, --max-running-requests and --cuda-graph-max-bs should be assigned the batch size

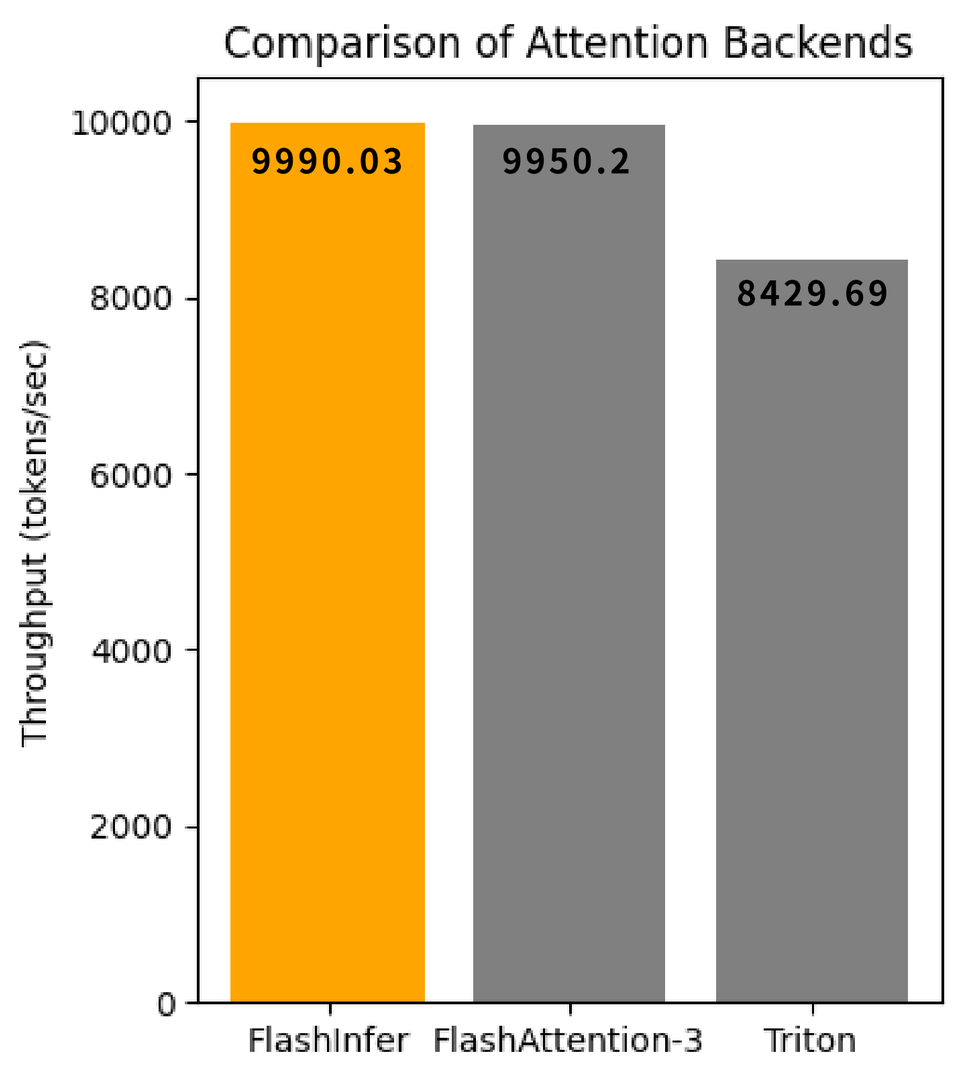
https://github.com/sgl-project/sglang/blob/main/scripts/playground/bench_speculative.py

Updating Our Baseline -Base 2-

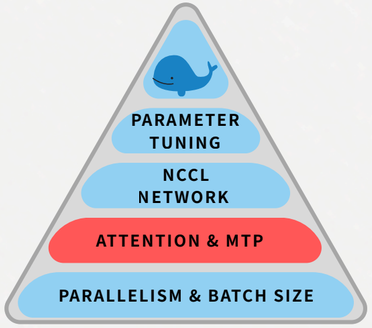


BS=1024, 10848.09(tok/s)

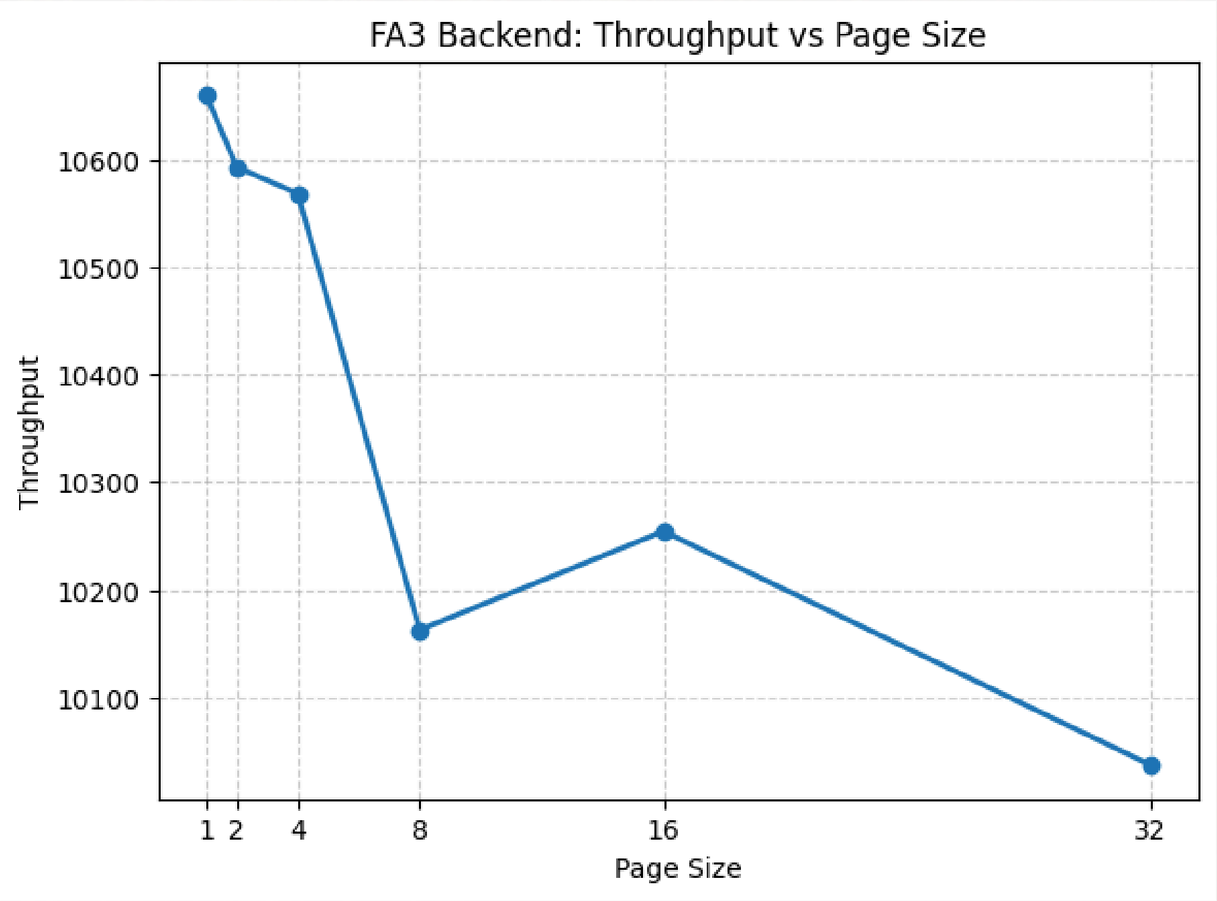
Attention Backend



- Flashinfer is optimized for inference and produced the highest throughput
- FlashAttention-3 was default
- Backends such as TRTLLM does not support Hopper GPU architechture



Page-size



Supporting matrix for different attention backends

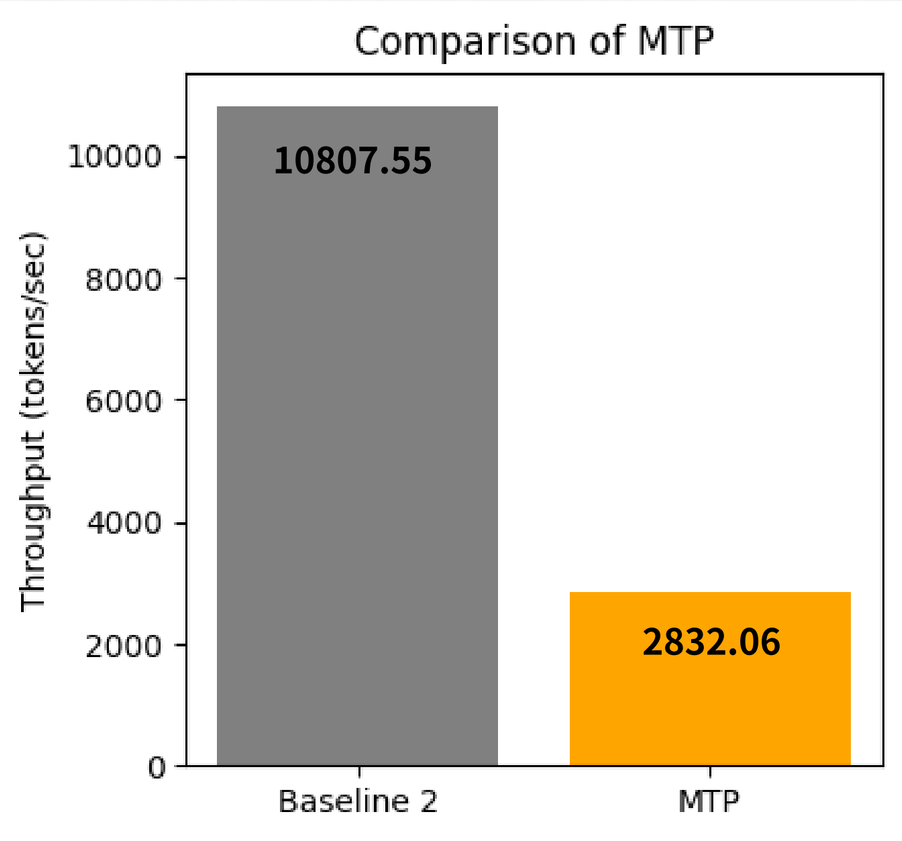
Backend	Page Size > 1	Spec Decoding	MLA	Sliding Window	MultiModal
FlashInfer	✗	✓	✓	✓	✓
FA3	✓	✓	✓	✓	✓
Triton	✗	✓	✓	✓	✗

https://docs.sglang.ai/advanced_features/attention_backend.html

Screenshot taken on Oct 9, actual website as since been recently updated

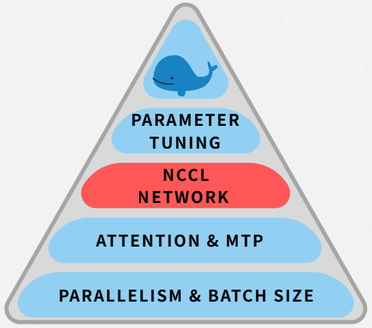
- FlashInfer does not support >1 page size, therefore FA3 was used
- We found that throughput drops as we increased page size

Multi Token Prediction



Hypothesis:

- After many tries, we discovered that we were required to decrease **--mem-fraction-static** from 0.8 down to 0.55 in order for it to work
- Due to that requirement, throughput was poor for any configuration of MTP parameters that we tried



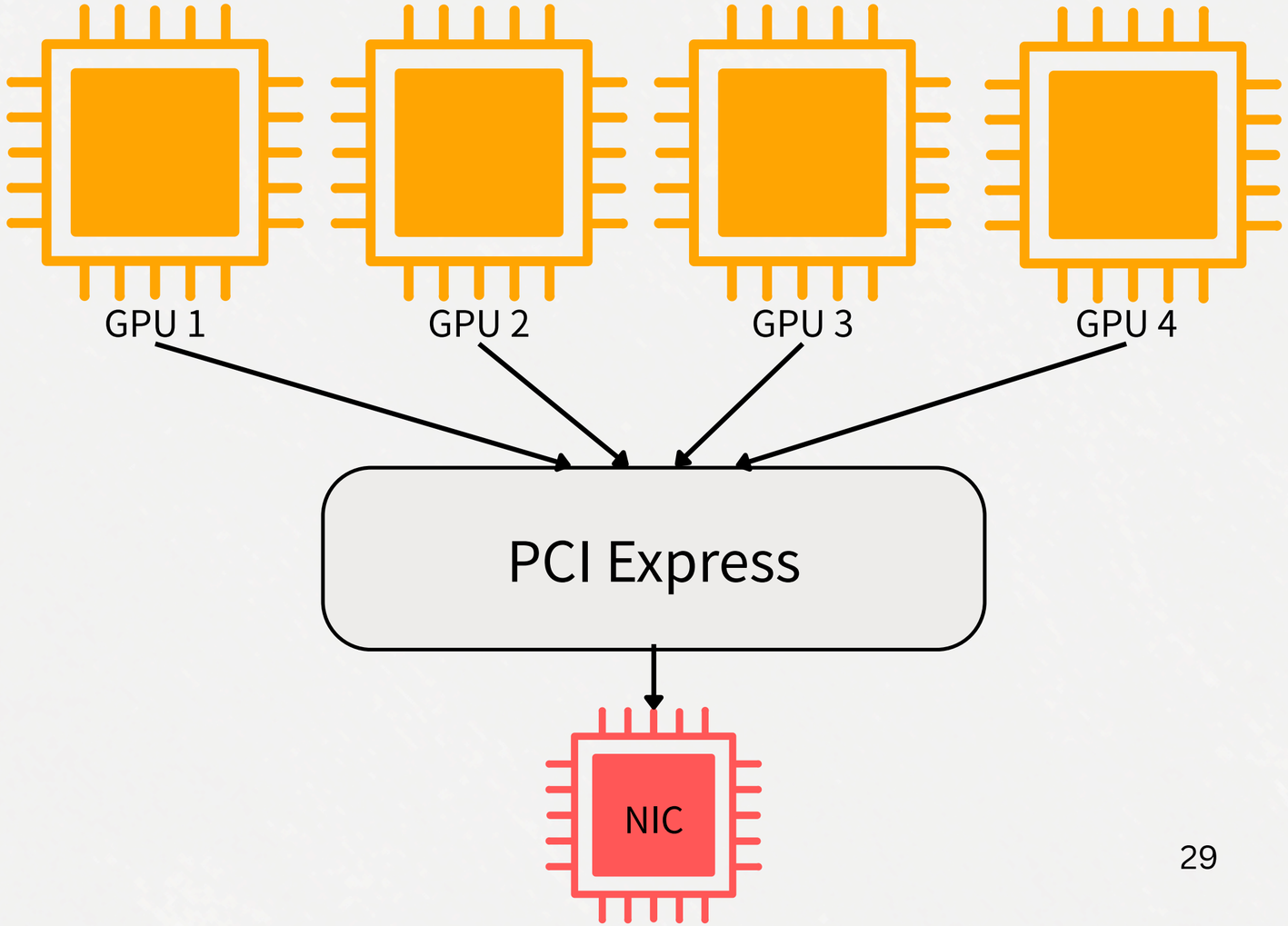
NCCL Tuning: GPUDirect RDMA

NVIDIA Collective Communication Library (NCCL)

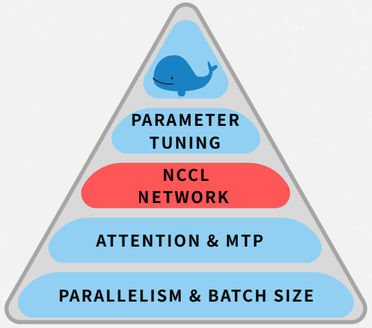
- Provide efficient multi-GPU and multi-node communication primitive
- Support GPU-to-GPU communication without through the CPU

GPUDirect RDMA

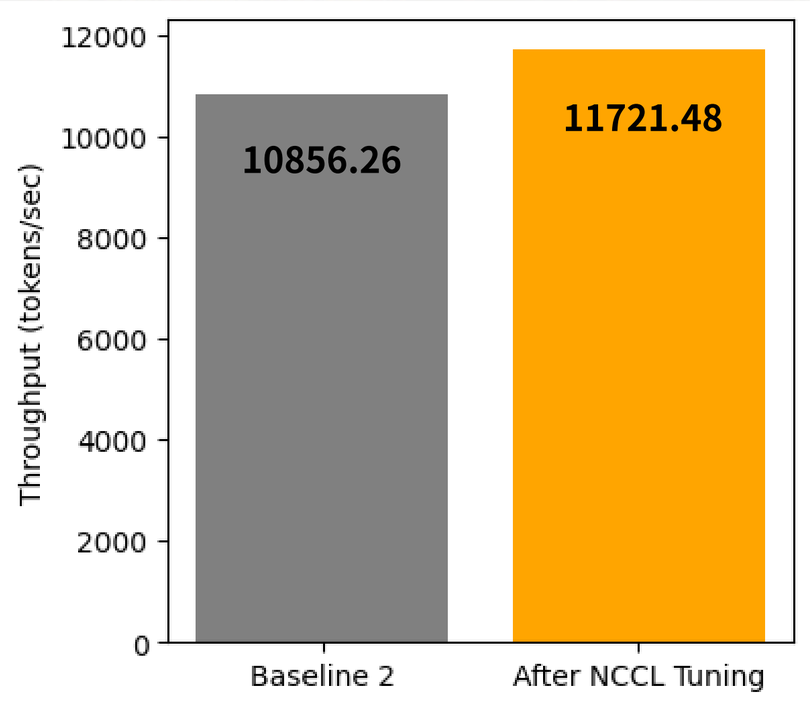
- Enables a direct path for data exchange between the GPU and network interfaces.



NCCL Tuning

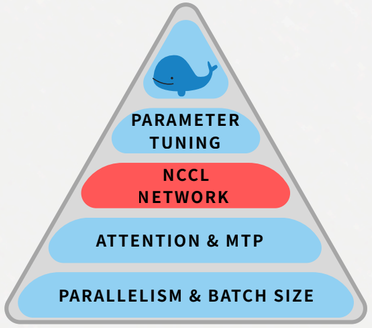


- We set env variables to tune NCCL, such as:
  - export NCCL_NET_GDR_LEVEL=SYS
  - export NCCL_NET_GDR_READ=1
  - ...




```
# Choose ONLY IB HCAs (adjust names to your node: mlx5_0..mlx5_11 are shown)
export NCCL_IB_HCA="mlx5_0,mlx5_1,mlx5_3,mlx5_4,mlx5_5,mlx5_6,mlx5_7,mlx5_9,mlx5_10,mlx5_11"
# Make NCCL stop trying to "fuse" different NIC types
export NCCL_NET_MERGE_LEVEL=LOC
# Prefer IB GID index 0 for IB (if your site uses a different index for RoCE/IB, set accordingly)
export NCCL_IB_GID_INDEX=0

export NCCL_NET_GDR_LEVEL=SYS
export NCCL_NET_GDR_READ=1
#export NCCL_PXN_DISABLE=1

export SGLANG_HACK_PD_DECODE_NUM_RESERVED_DECODE_TOKENS=2
export SGLANG_SET_CPU_AFFINITY=1
export SGL_ENABLE_JIT_DEEPGEMM=1
export MC_TE_METRIC=true

export NCCL_CROSS_NIC=0
export NCCL_P2P_NVL_DISABLE=0
export NCCL_SHM_DISABLE=0
export NCCL_IB_TC=0

export NCCL_ALGO=Tree,Ring,CollnetDirect,CollnetChain,NVLS,NVLSTree
export NCCL_NVLS_ENABLE=1
export NCCL_P2P_LEVEL=NVL
```

How we specified NCCL_IB_HCA

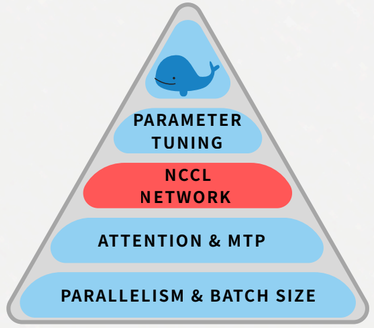



```
~/ochi/solve/aspire/r1/exp_tsune_00f$ grep -E 'hca_id|link_layer' ibv_devinfo
hca_id: mlx5_0
                link_layer:             InfiniBand
hca_id: mlx5_1
                link_layer:             InfiniBand
hca_id: mlx5_2
                link_layer:             Ethernet
hca_id: mlx5_3
                link_layer:             InfiniBand
hca_id: mlx5_4
                link_layer:             InfiniBand
hca_id: mlx5_5
                link_layer:             InfiniBand
hca_id: mlx5_6
                link_layer:             InfiniBand
hca_id: mlx5_7
                link_layer:             InfiniBand
hca_id: mlx5_8
                link_layer:             Ethernet
hca_id: mlx5_9
                link_layer:             InfiniBand
hca_id: mlx5_10
                link_layer:             InfiniBand
hca_id: mlx5_11
                link_layer:             InfiniBand
```

The **NCCL_IB_HCA** specifies which interfaces to use for communication.

Use only **InfiniBand devices** to force the node to use only InfiniBand.

```
export NCCL_IB_HCA="mlx5_0,mlx5_1,mlx5_3,mlx5_4,mlx5_5,mlx5_6,mlx5_7,mlx5_9,mlx5_10,mlx5_11"
```



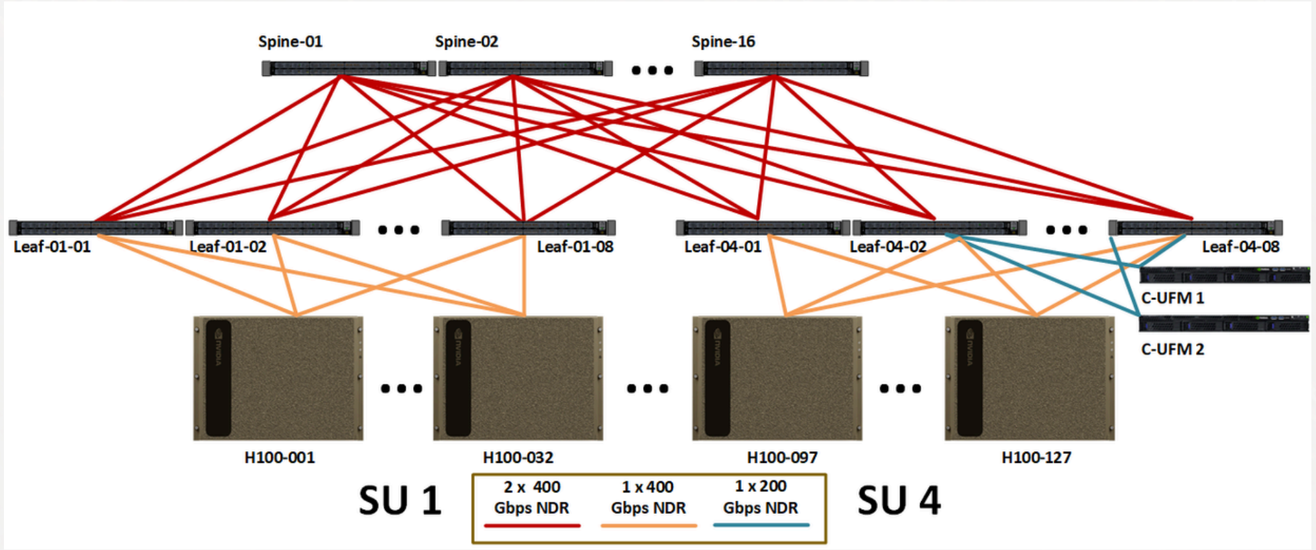
Variance between Nodes

DGX SuperPOD Architecture

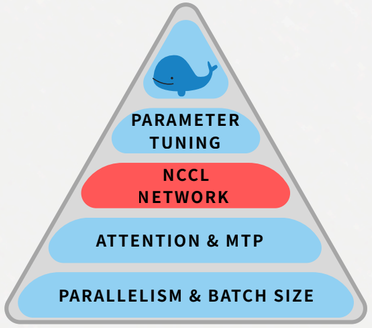
- Ensures optimal traffic routing and that performance is consistent across all portions of the fabric.

HOWEVER, in our experiments:

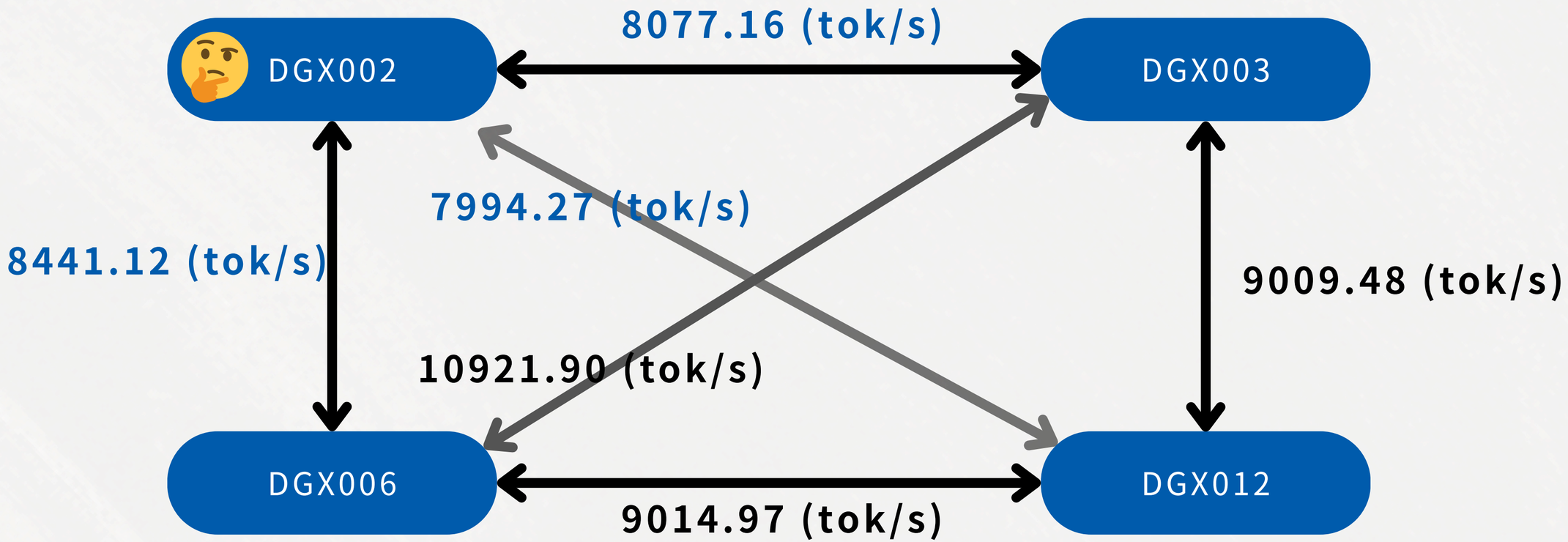
- Throughput of the same jobscript varies by around 1x~1.2x when executed multiple times
- We hypothesize that this is due to the allocation of different node combinations



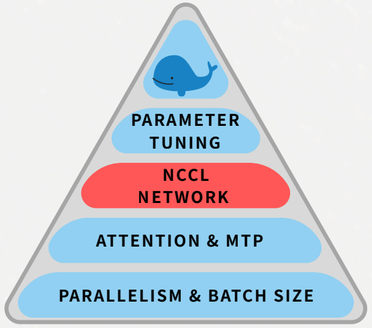
https://docs.nvidia.com/dgx-superpod/reference-architecture-scalable-infrastructure-h100/latest/network-fabrics.html



Variance between Nodes



DGX002 seems to be problematic



Patching to SGLang

- We increased warmup by:
  - increasing num_prompts to 512
  - increasing input, output lengths to 4096, 256 respectively
- This is clearly **non-destructive to the generation quality**

```
warmup_requests = sample_random_requests(
    input_len=256,
    output_len=16,
    num_prompts=min(bench_args.num_prompts, 16),
    range_ratio=1.0,
    tokenizer=tokenizer,
    dataset_path=bench_args.dataset_path,
)
```

sglang/benchmark_offline_throughput.py

PATCH

```
warmup_requests = sample_random_requests(
    input_len=4096,
    output_len=256,
    num_prompts=min(bench_args.num_prompts, 512),
    range_ratio=1.0,
    tokenizer=tokenizer,
    dataset_path=bench_args.dataset_path,
)
```

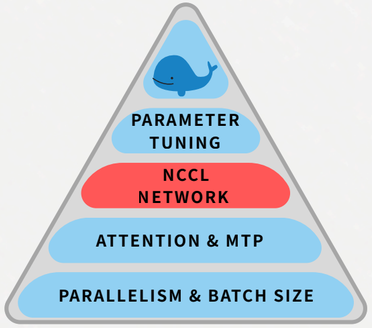
PATCHED sglang/benchmark_offline_throughput.py

- We also used input_requests (i.e. benchmark dataset) for warmup instead of warmup_requests, and the throughput increased even further

```
# Warm up
if not bench_args.skip_warmup:
    logging.info("\nWarmup...")
    throughput_test_once(
        backend_name=bench_args.backend,
        backend=backend,
        reqs=warmup_requests,
        ignore_eos=not bench_args.disable_ignore_eos,
        extra_request_body=extra_request_body,
        profile=False,
    )
    time.sleep(0.5)
```

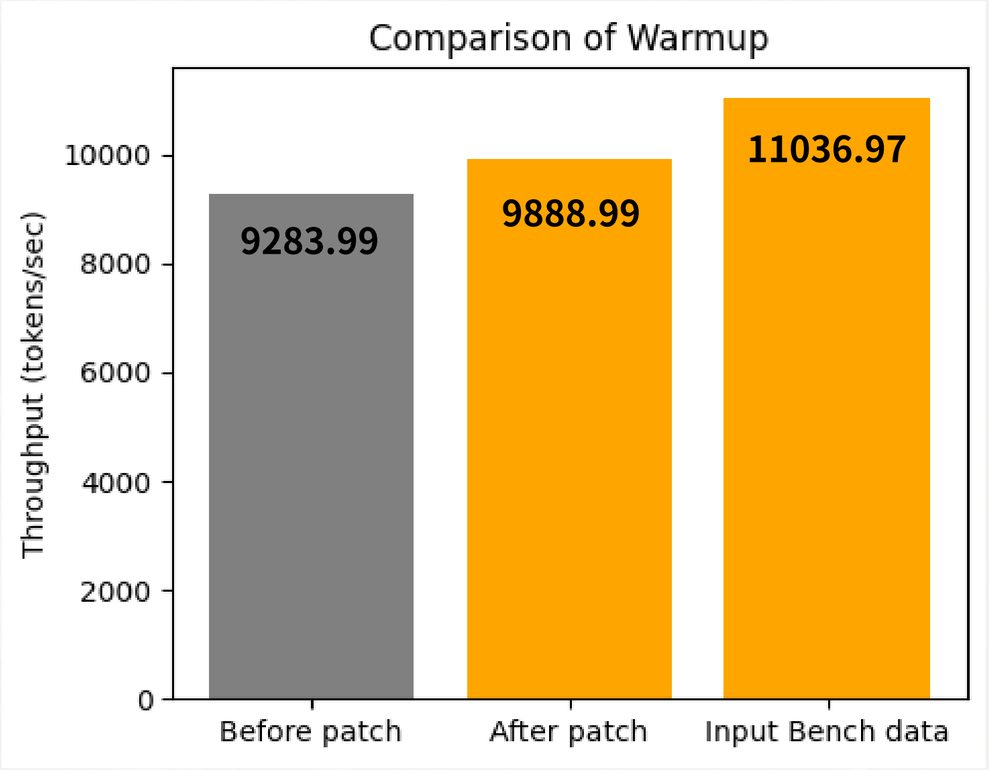
→ input_requests

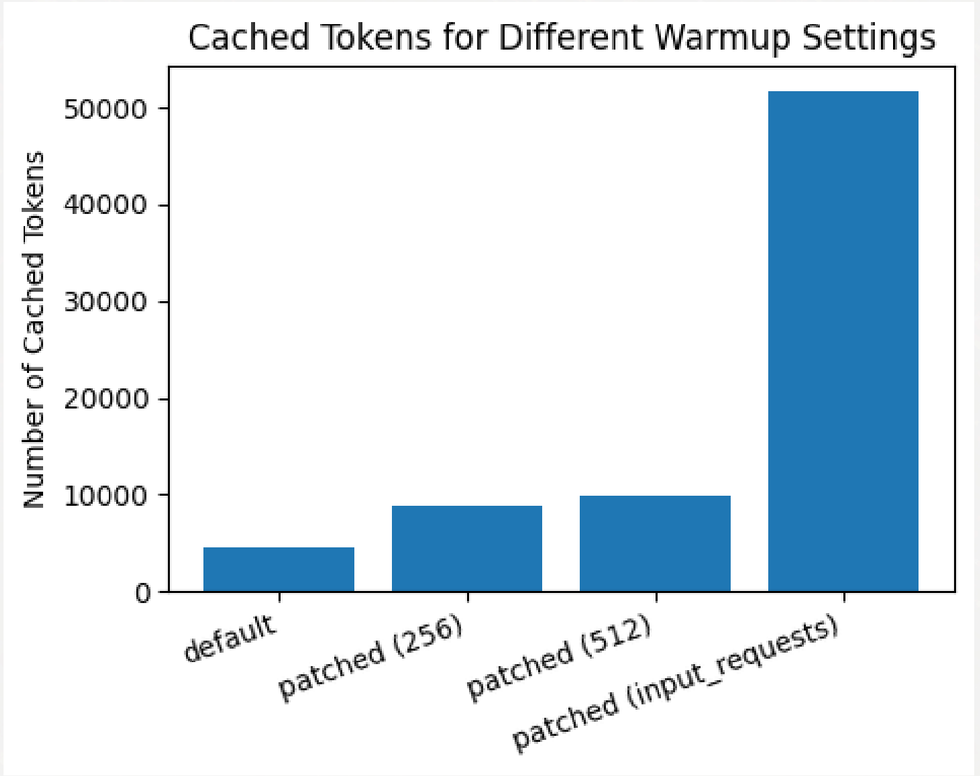
PATCHED sglang/benchmark_offline_throughput.py

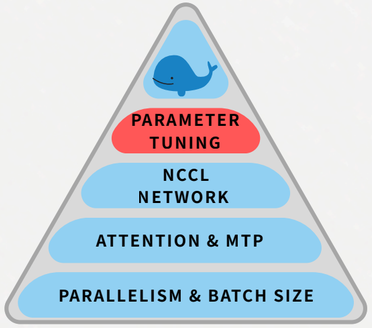


Patching to SGLang

- By warming up compute nodes with the same dataset, its KV cache can be prefilled before the actual benchmark, leading to faster inference

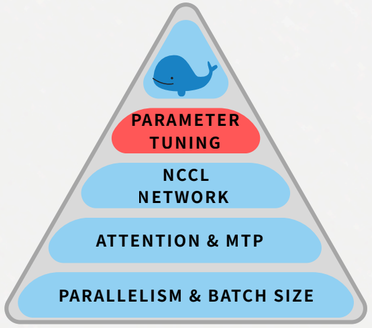






Other parameters

Changes	Description	Baseline	throughput (tokens/s)
--disable-radix-cache	Turns off RadixAttention prefix caching	10087.56 10399.40 10803.40	10807.55 10832.35 10904.22
--disable-overlap-schedule	Stops overlapping CPU scheduling work	10004.89 10855.00 10791.86	10806.71 10902.25 10822.39
--grammar-backend xgrammar	Offers faster JSON decoding performance	10004.93	7941.49



Final tuning

- At the end of optimization, we tuned --mem-fraction-static
- It specifies the GPU memory allocated to model weights and KV cache

--mem-fraction-static	Max GPU Memory Used (TB)	Throughput
0.80	1.14	9239.17
0.88	1.23	10152.85
0.89	OOM	-

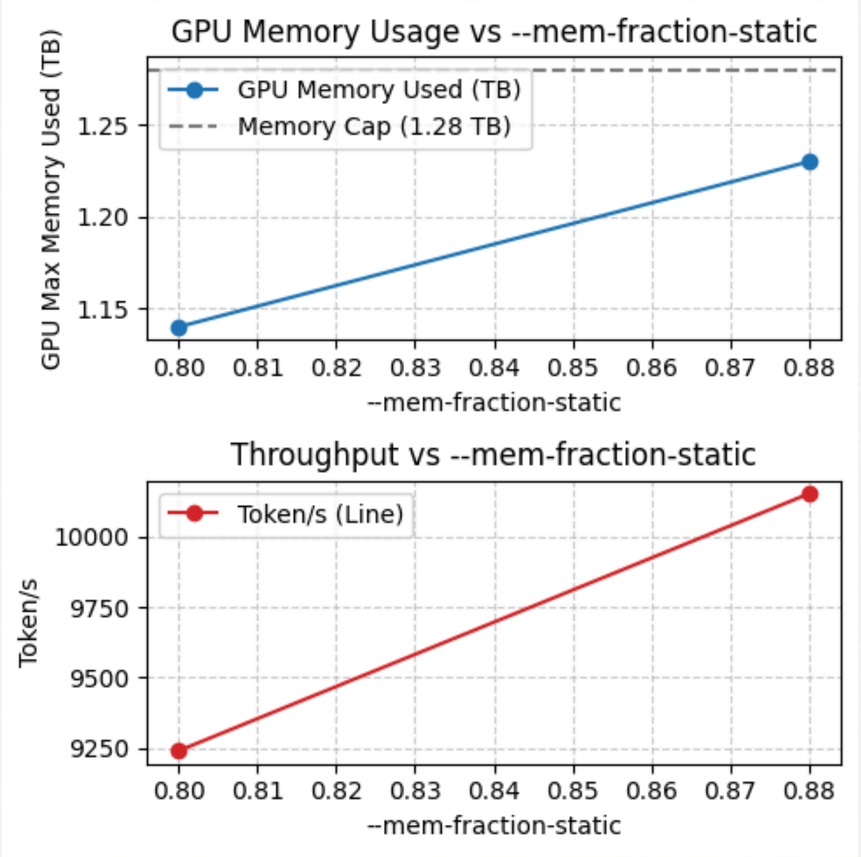
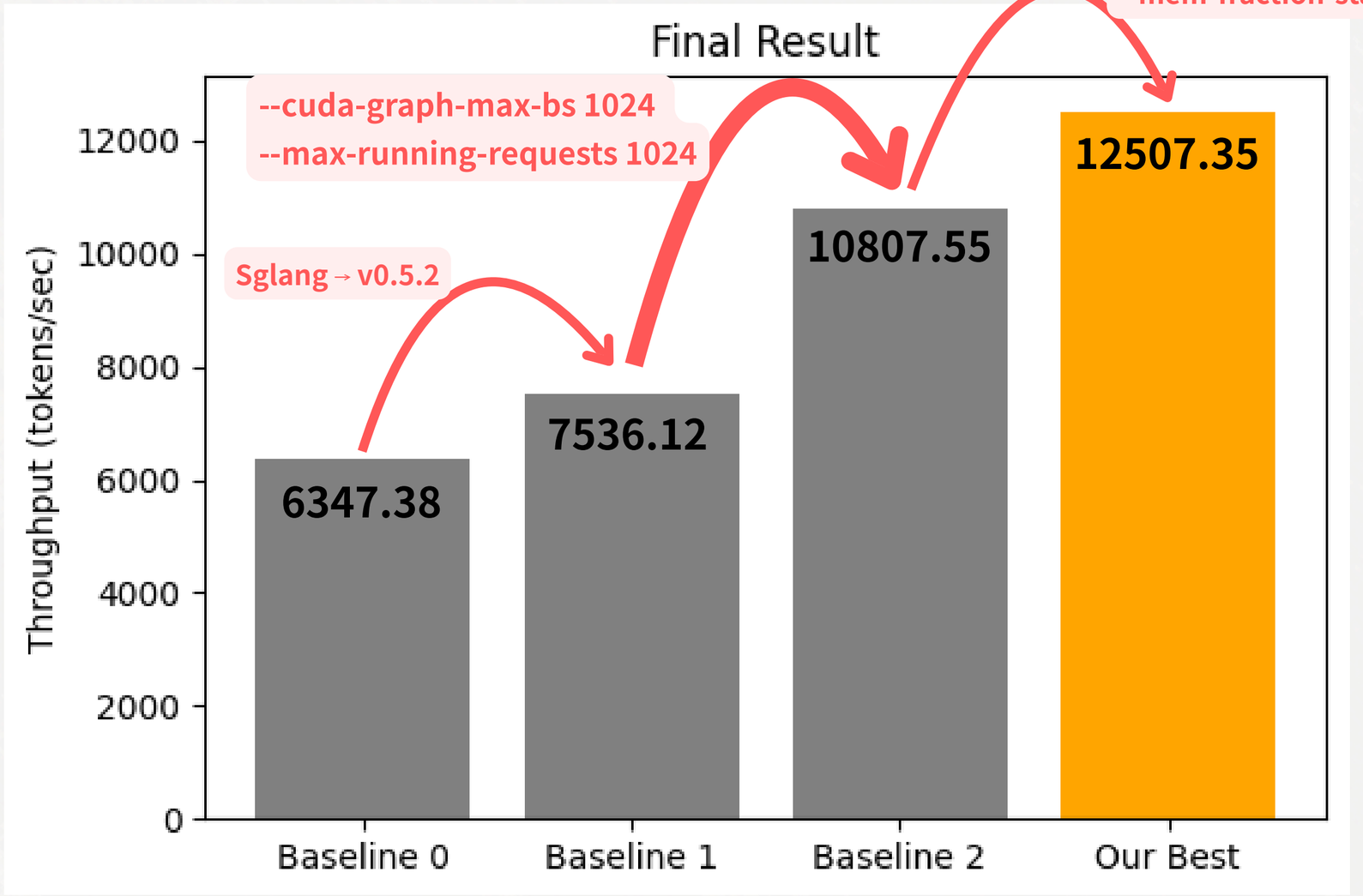


Table of Contents

Summary of Results



Acknowledgements

This work was supported by RIKEN R-CCS.
The computer resource offered under the category of General Projects by the Research Institute for Information Technology, Kyushu University was used for training purposes.

The End

Thank You For Listening

APAC HPC-AI Challenge

KBB