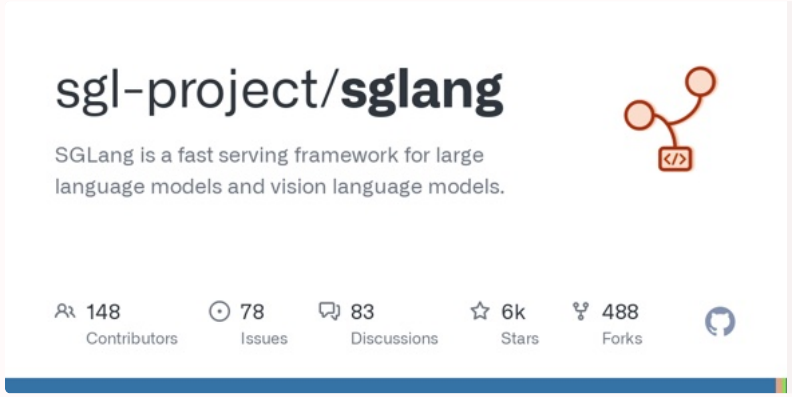
SGLang Performance Optimization

Yineng Zhang @ Baseten

About SGLang



GitHub

GitHub - sgl-project/sglang: SGLang is a fast serving framewor...

SGLang is a fast serving framework for large language models and vision language models. - sgl-project/sglang

SGLang is a fast serving framework for large language models and vision language models.

SGLang is an LLM Inference Engine incubated by **LMSYS Org**. It includes both **frontend** and **backend** components. The backend functionality is similar to TensorRT LLM and vLLM. Among fully open-source LLM Inference Engines, SGLang currently achieves the **SOTA performance**.

Meanwhile, its **lightweight and customizable design** has attracted adoption from both big companies and startups.

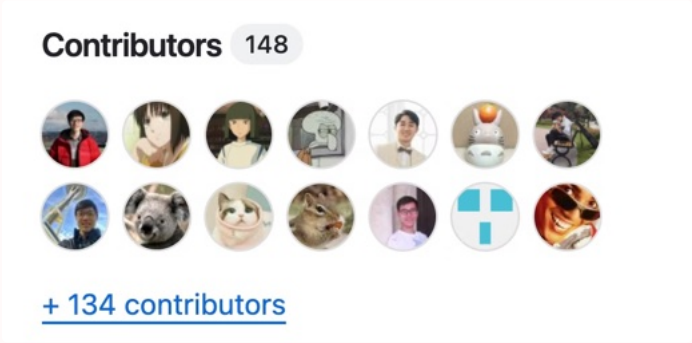
About SGLang Team

SGLang Team is led by Lianmin Zheng and Ying Sheng.

Core Team Members

Lianmin Zheng(Creator), Ying Sheng(Creator), Liangsheng Yin(Creator), Yineng Zhang, Byron Hsu, Ke Bao, Chenyang Zhao

Contributors (100+)



Special Acknowledgment

Zihao Ye(FlashInfer), Henry Xiao(AMD), Li Zhang(LMDeploy), Han Lyu(LMDeploy), Jerry Zhang(TorchAO), Mark Saroufim(TorchAO)

About me

Yineng Zhang

Experience:

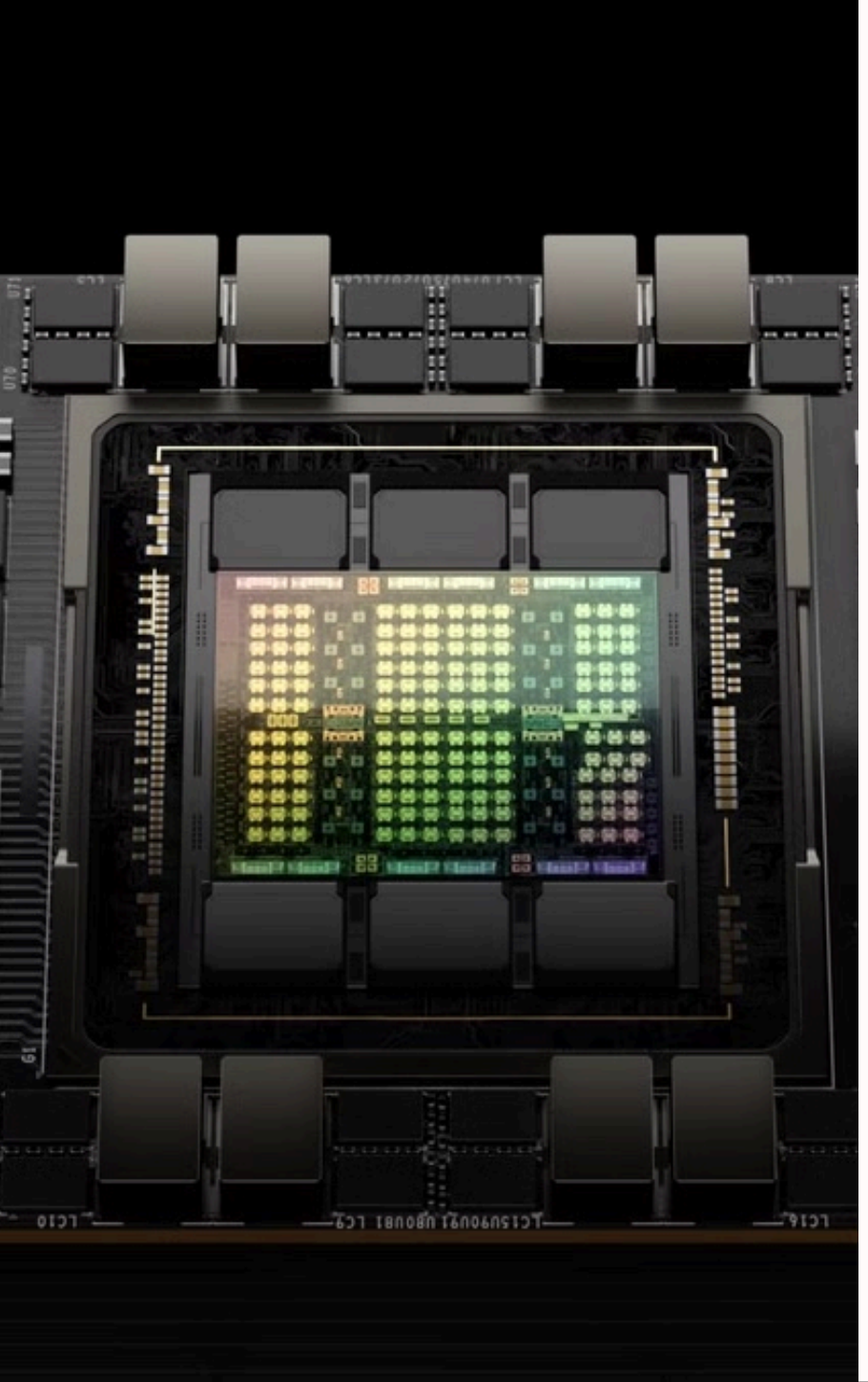
- 2024.9-now Software Engineer on the Model Performance Team at **Baseten**
  (LLM Inference **https://www.baseten.co/blog/**)
- 2024.7-now Team Member of SGLang Team and **LMSYS Org**
- 2021.8-2024.7 Software Engineer on the Machine Learning Engine Group at **Meituan**
  (CTR prediction for search recommendations and LLM Inference)
- 2020.6-2021.8 Software Engineer on Baidu Speech at **Baidu**

Email: me@zhyncs.com
GitHub: **https://github.com/zhyncs**
LinkedIn: **https://linkedin.com/in/zhyncs**
Twitter: **https://x.com/zhyncs42**

SGLang Milestone

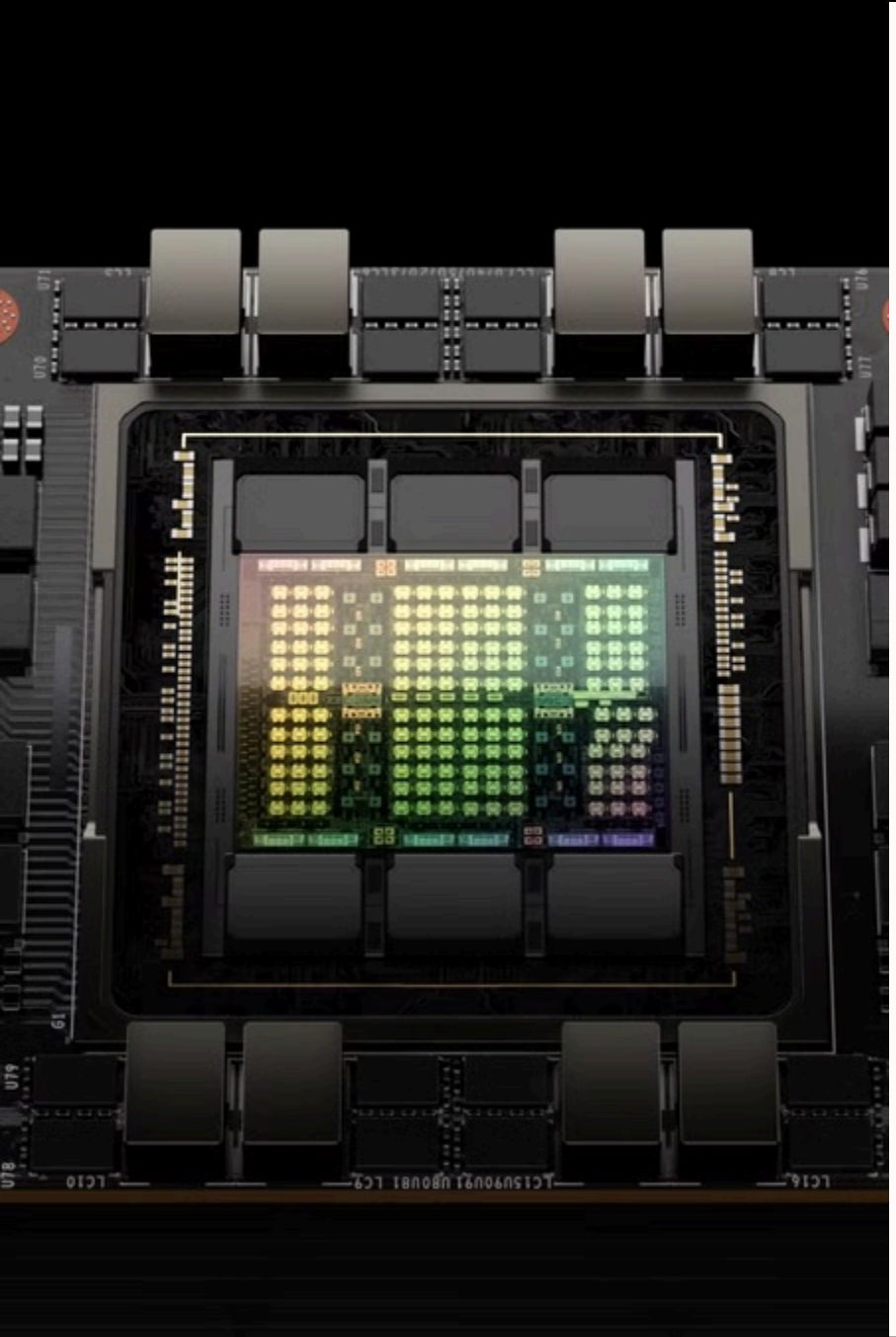
- Aug - Dec 2023: Initial Motivation: the "programming LLM" paradigm https://arxiv.org/abs/2312.07104
- Jan 2024: The first to support LLM Prefix Cache https://lmsys.org/blog/2024-01-17-sglang/
- Feb 2024: The first to support LLM Constrained Decoding https://lmsys.org/blog/2024-02-05-compressed-fsm/
- July 2024: Fully-functional LLM Inference Engine with impressive performance https://lmsys.org/blog/2024-07-25-sglang-llama3/
- Sep 2024: Proceed with ongoing feature enhancements and performance improvements https://lmsys.org/blog/2024-09-04-sglang-v0-3/

Coming Soon: SGLang v0.4 (better performance and stability)



SGLang Performance Optimization Agenda

1. CPU Overlap Optimization
2. FlashInfer Optimization
3. TurboMind GEMM Optimization

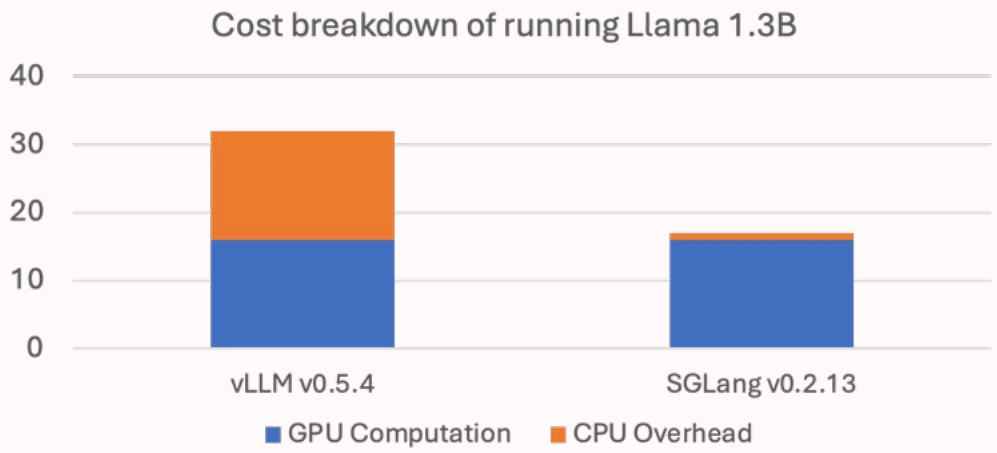


SGLang Performance Optimization

CPU Overlap Optimization

This enhancement minimizes SGLang's CPU overhead from previous implementations, and is now available in the latest release.

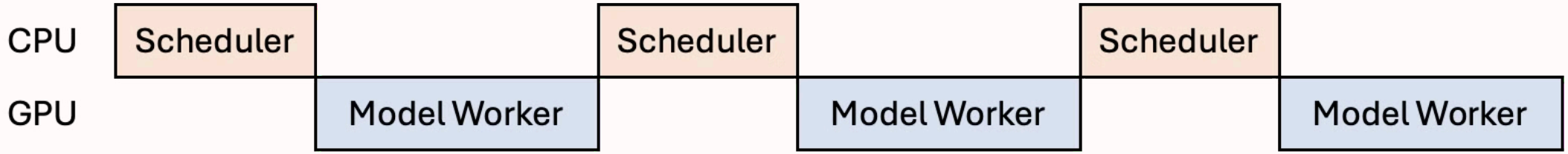
CPU overhead hiding



Source: https://mlsys.wuklab.io/posts/scheduling_overhead/

An unoptimized inference engine can waste more than 50% time on CPU scheduling.

CPU overhead



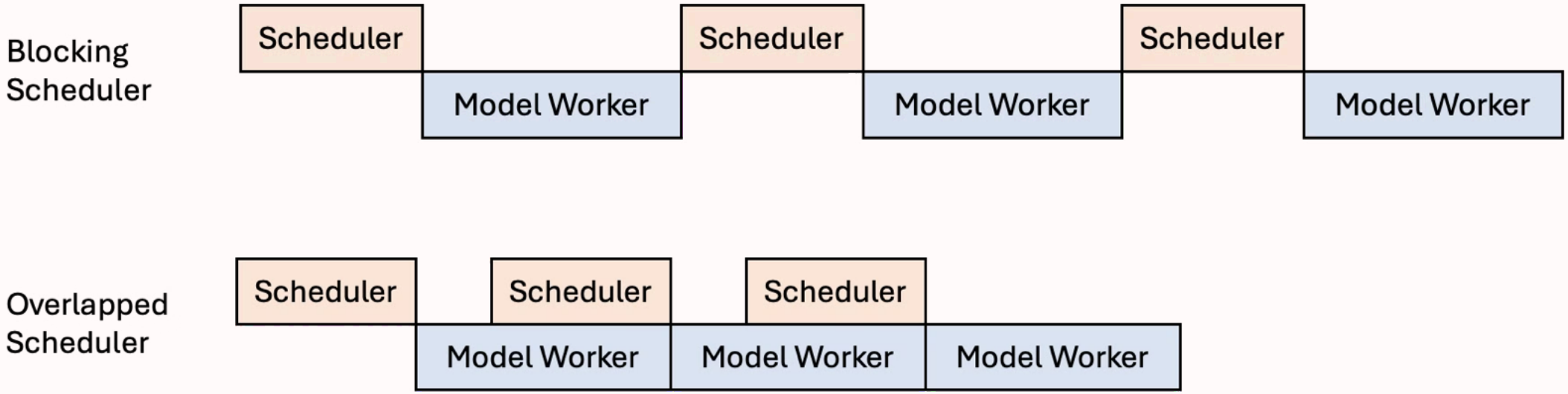
Jobs of the CPU scheduler

- Receives input messages from the user
- Processes results from the model worker
- Checks the stop conditions
- Runs prefix matching and request reorder
- Allocates memory for the next batch

Pseudo code (every line is blocking)

```
while True:
    recv_reqs = recv_requests()
    process_input_requests(recv_reqs)
    batch = get_next_batch_to_run()
    result = run_batch(batch)
    process_batch_result(batch , result)
```

Overlapped scheduler



Implementation in the SGLang

SGLang implements two scheduler loops (#1738, #1677, #1687). They share almost all other functions.

You can turn on the overlap version with `--enable-overlap`

Normal version

```
while True:
    recv_reqs = recv_requests()
    process_input_requests(recv_reqs)
    batch = get_next_batch_to_run()
    result = run_batch(batch)
    process_batch_result(batch , result)
```

Overlap version

```
last_batch = None
while True:
    recv_reqs = recv_requests()
    process_input_requests(recv_reqs)
    batch = get_next_batch_to_run()
    result = run_batch(batch)
    result_queue.put((batch, result))
    if last_batch is not None:
        tmp_batch, tmp_result = result_queue.get()
        process_batch_result(tmp_batch, tmp_result)
    last_batch = batch.copy()
```

Key implementation challenges

- How to resolve the dependency?
- How to share as much code as possible for overlap and non-overlap version?
- How to workaround Python GIL?

Resolve the dependency

Idea: delay the finish condition check

We can assume a request did not finish and immediately run it in the next decoding batch

We resolve the dependency by paying the overhead of decoding one more useless token, but this is negligible

Reuse code between overlap and non-overlapped versions

Idea: introduce a new tensor type “Future tokens”

The model worker returns a future of next token ids, making the `run_batch` call non-blocking.

Most scheduler operations only need the shape of the tokens and do not need the real values, so they directly operate the future objects without knowing the values

```
last_batch = None
while True:
    recv_reqs = recv_requests()
    process_input_requests(recv_reqs)
    batch = get_next_batch_to_run()
    result = run_batch(batch)
    result_queue.put((batch, result))
    if last_batch is not None:
        tmp_batch, tmp_result = result_queue.get()
        process_batch_result(tmp_batch, tmp_result)
    last_batch = batch.copy()
```

Workaround Python GIL (Work in progress)

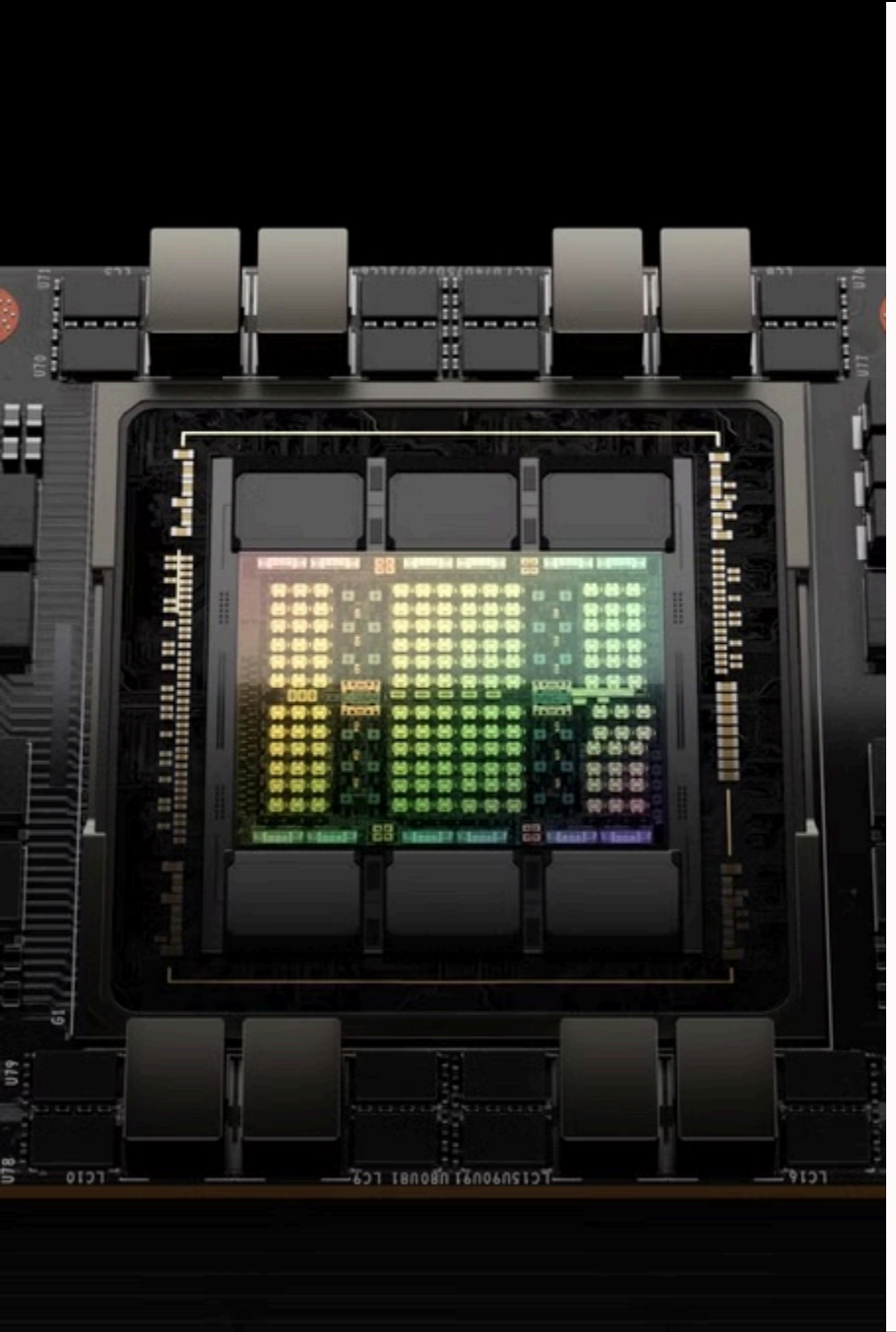
- **Idea 1**: Try Python 3.13 which can remove GIL
- **Idea 2**: Use multiple processes, but need make the serialization very fast
- **Current progress**: Still a single thread, but use the free CPU cycles after CUDA Graph launch

Next steps

- Try Python 3.13 without GIL
- Implement the multi process version
- Try some faster HTTP server

Expect about 10-20% base performance improvement across all workloads in the next release SGLang v0.4

Benchmark results across various scenarios will be released at v0.4 launch



SGLang Performance Optimization

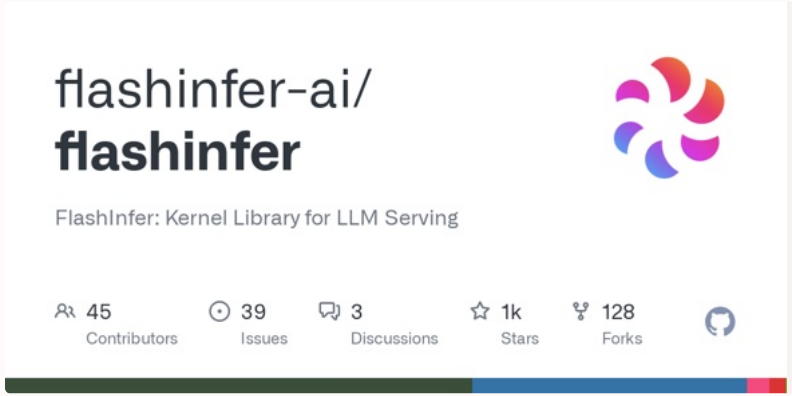
FlashInfer Optimization

SGLang has integrated FlashInfer as its default backend since v0.2. Let's explore the upcoming optimizations that FlashInfer plans to introduce.

SGLang has conducted integration and performance benchmark in advance, with an upcoming release planned.

About FlashInfer

FlashInfer is a library for Large Language Models that provides high-performance implementation of LLM GPU kernels such as FlashAttention, SparseAttention, PageAttention, Sampling, and more. FlashInfer focuses on LLM serving and inference, and delivers state-of-the-art performance across diverse scenarios.



GitHub

GitHub - flashinfer-ai/flashinfer: FlashInfer: Kernel Library for L...

FlashInfer: Kernel Library for LLM Serving. Contribute to flashinfer-ai/flashinfer development by creating an account on GitHub.

SGLang integrates FlashInfer's Attention, Sampling, Norm and Activation kernels.

A specialized presentation on FlashInfer is expected to be shared by **creator** Zihao Ye in January 2025 in GPU MODE.

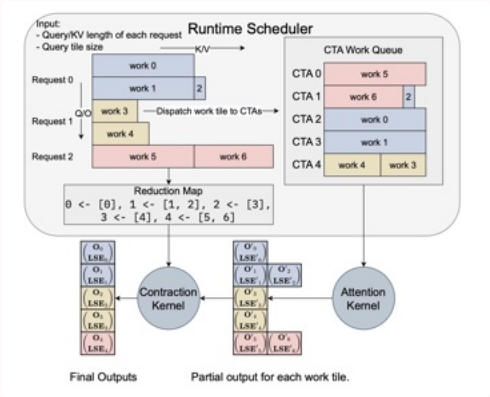
JIT

It's inspired by **FlexAttention**. FlashInfer designed a customizable CUDA template and a JIT compiler that takes the attention variant specification as input and generates the optimized kernel code, such as CUDA or CUTLASS. **Main branch supports JIT while keeping AOT, hasn't released a new version yet.**

FlashAttention 3

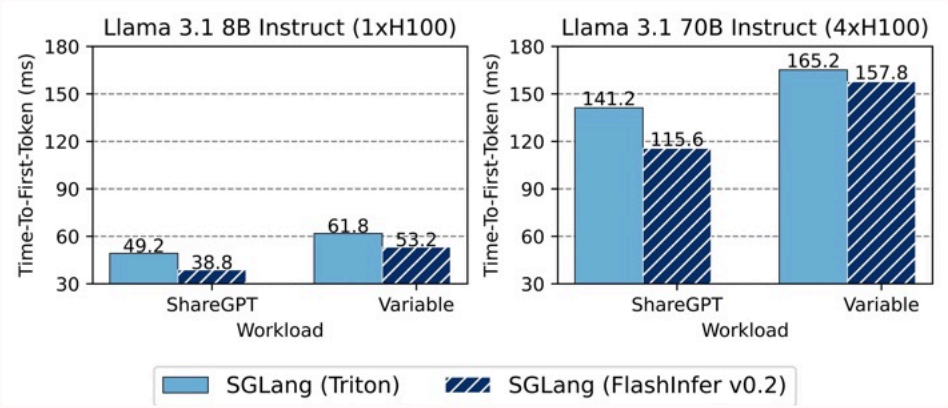
In addition to the **original FlashAttention 3** optimizations, **FlashInfer introduces support for paged KV cache**, a feature absent in the original implementation.**This part of the implementation is already complete, it has not been made public yet and is expected to be landed soon.**

Stream K Scheduler

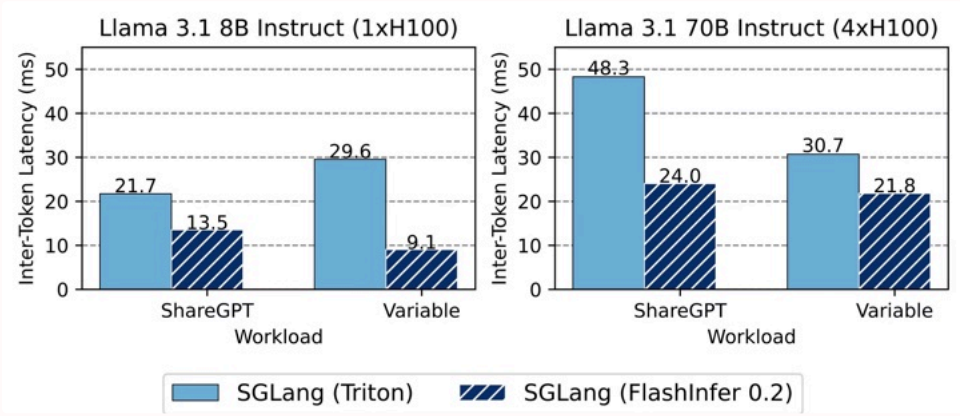


It's inspired by **Stream K**, aims to minimize the idle time of SMs by distributing the workload evenly across all SMs. The algorithm takes the sequence length information of the query/output and key/value dimensions as input, and outputs the mapping between work and CTAs, and the index mapping between partial and final outputs. **The implementation is complete, it is expected to be landed soon.** In scenarios where the variance of the input distribution is more uneven, the benefits are more significant.

Benchmark Results (vs SGLang Triton)

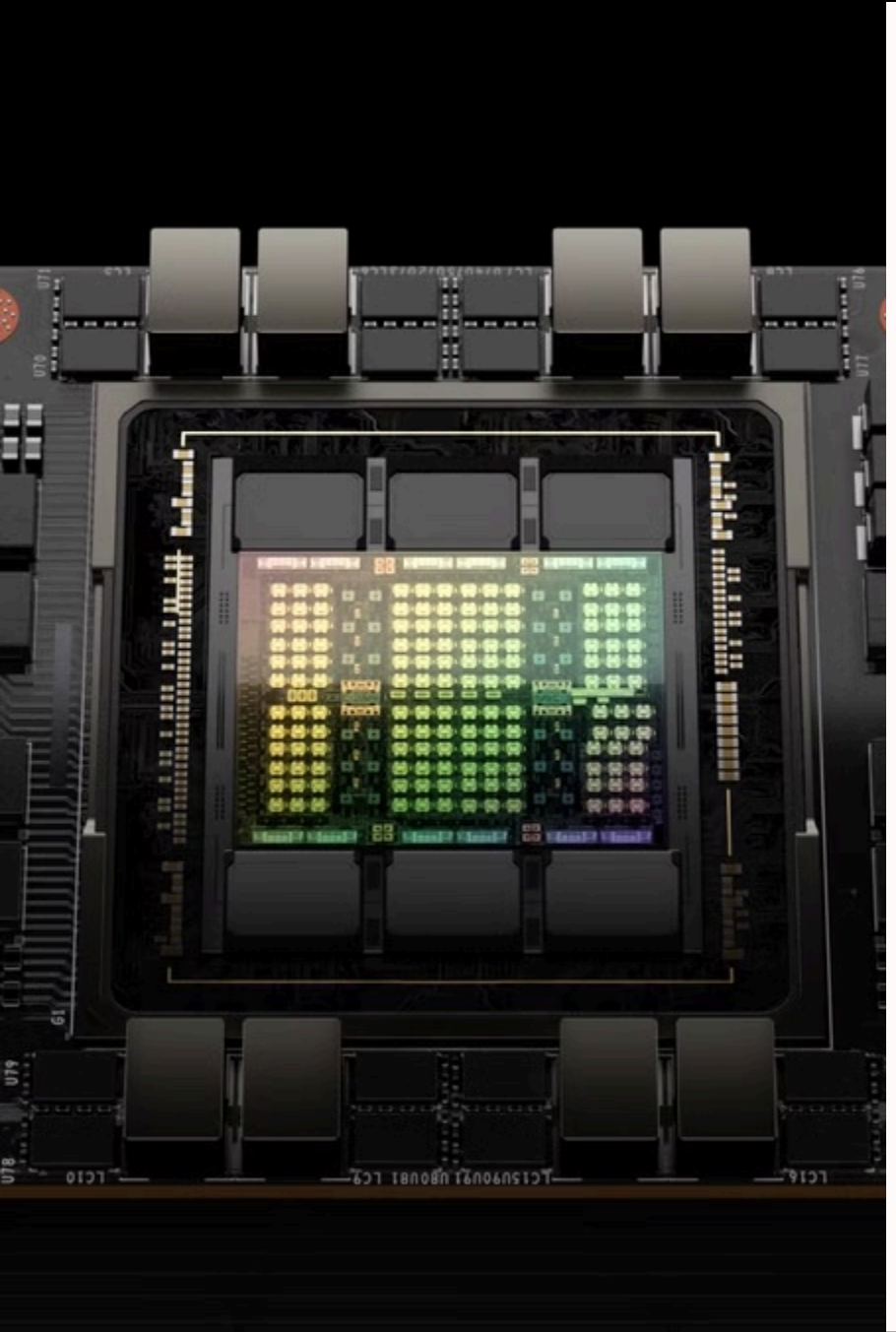


TTFT in ShareGPT and Random scenario



ITL in ShareGPT and Random scenario

My work primarily focuses on integrating and validating the upcoming FlashInfer v0.2 release with SGLang, including performance benchmark. Join FlashInfer Slack channel **https://t.ly/tDs7r** for the detailed technical discussion with FlashInfer's **creator** **Zihao Ye**.



SGLang Performance Optimization

TurboMind GEMM Optimization

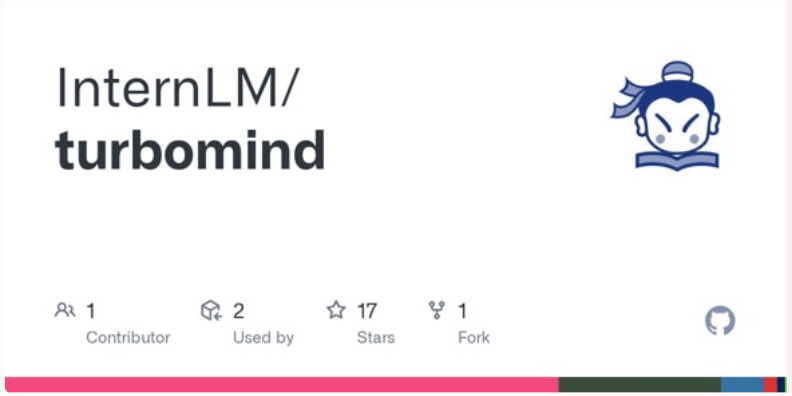
SGLang plans to integrate TurboMind to enhance the performance of current quantization and MoE.

About TurboMind

LMDeploy is a toolkit for compressing, deploying, and serving LLMs. It's developed by Shanghai AI Lab.

The initial release in July 2023 was built upon a modified version of FasterTransformer. Subsequent iterations featured a complete rewrite of the Attention and request scheduling, significantly enhancing the inference performance.

TurboMind is a specialized GEMM optimization component that was originally part of LMDeploy and has been extracted and released as open-source. It primarily focuses on accelerating linear operations, including quantization and MoE.

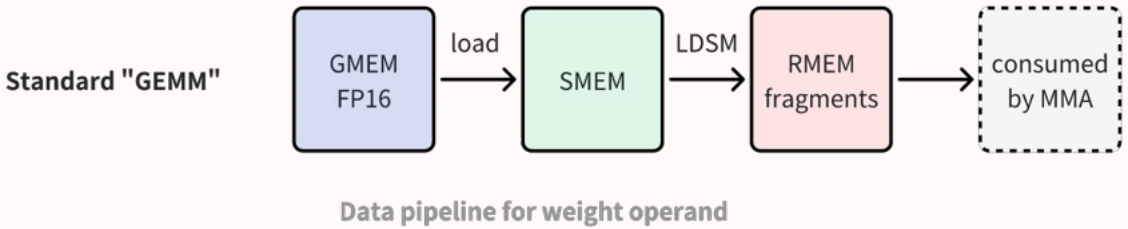


GitHub

GitHub - InternLM/turbomind

Contribute to InternLM/turbomind development by creating an account on GitHub.

GEMM operands in the tensor core era



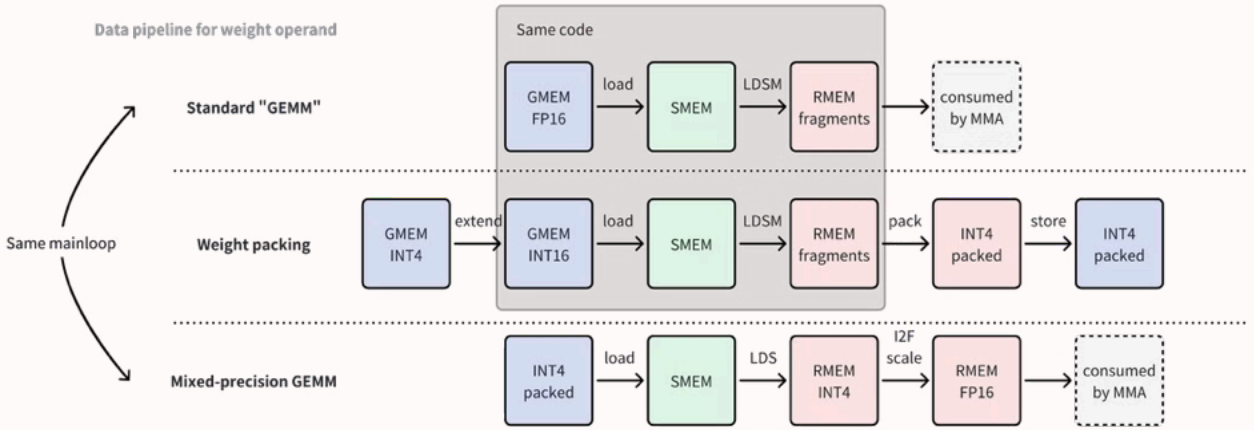
Data pipeline for weight operand

1. Load the operand to from global memory to shared memory (GMEM -> SMEM)
2. Load matrix fragments from shared memory to registers (SMEM -> RMEM)
3. Feed the tensor core MMA instruction with the fragments

Step 2 relies on instructions such as LDSM to effectively transfrom the col/row-major layout to the layouts required by MMA instructions.

However, when the operand's bit-width is different from Tensor Core's input bit-width (e.g. INT4/INT8 vs FP16) the loaded data will be misaligned because the instruction is not designed to handle such mapping)

Packing the weights

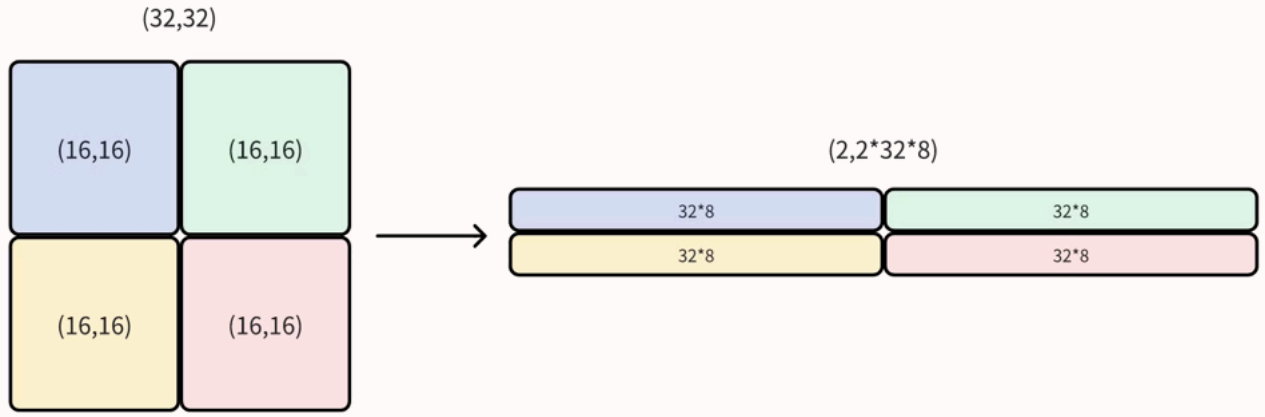


Instead of designing complex weight layouts to make the quantized weights compatible to specific tensor core instructions, we reuse the vender provided instructions like LDSM.

1. Bit-extend the 4-bit weight to 16-bit
2. Load the weight fragments with the same data pipeline as in "standard" GEMM
3. Bit-unextend 16-bit data to 4-bit (with additional reordering)
4. Each warp store the packed fragment back to GMEM

Operand A of mma.m16n8k16

For exmaple, 32×32 matrix can be view as 2×2 16x16 slices, each 16x16 (strided) tile is then packed into a contiguous 32x8 tile. The 2D packed shape is (2, 2*32*8) = (2,512). The transfromation that maps the original (m,n) coordinate to the packed shape is (m/16, n*16)

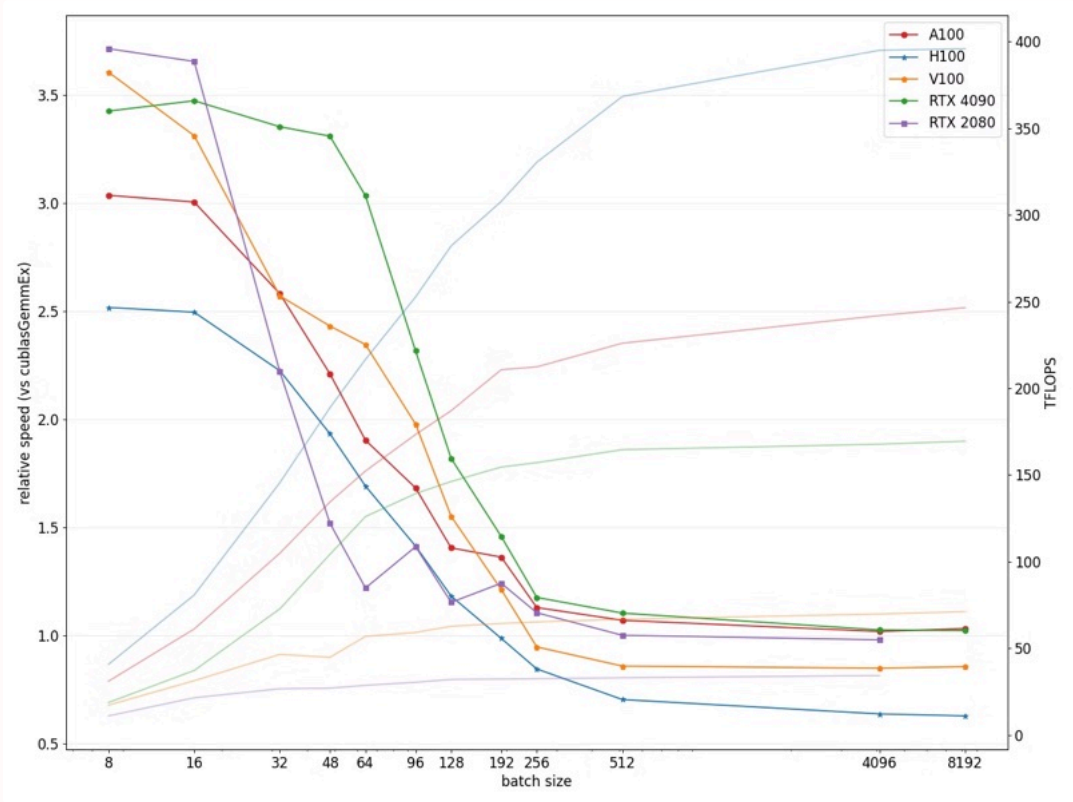


Advantages

- Saved layout directly maps to tensor core instructions
- Can be loaded naively without bank-conflicts
- Swizzling free, lower pressure on register file
- Adapts to any layout required by MMA instructions (even undefined)
- Adapts to any power-of-2 bit-width
- Number of fragments to pack is configurable

FP16xINT4 (asymmetrical, group_size=128, 4.25 actual bits) comparison with FP16xFP16 `cublasGemmEx`, average on 32 weight shapes from 8 LLMs from 7B to 72B

- A100, RTX 4090: never slower than FP16xFP16
- V100, RTX 2080: wins within reasonable decoding batch size (and faster than dequant+FP16xF16 fallback for reasonable prefill seqlen)
- H100: TMA+WGMMA optimizations on the way



Community users and contributors













One More Thing

SGLang Slack: **https://t.ly/AqR62** (**Please join the channel for the detailed technical discussion**)

Bi-Weekly Development Meeting: **https://t.ly/8YKlf**

Documentation: **https://sgl-project.github.io/** (support Jupyter Notebook)

Learning Materials: **https://github.com/sgl-project/sgl-learning-materials/**

2024 Q4 Development Roadmap: **https://github.com/sgl-project/sglang/issues/1487**

(Highlight: Speculative Decoding, Long Context Optimization, Disaggregated Prefill and Decoding etc.)

Q&A