LMSYS Online Meetup:
Efficient LLM deployment and serving

Agenda

16:00 - 16:45 SGLang updates: CPU overhead hiding, faster JSON decoding, MLA
16:50 - 17:35 FlashInfer updates: Kernel generation, JIT interface
17:40 - 18:30 MLC-LLM updates: Universal deployment, low-latency serving

Organizers:

Ying Sheng, Byron Hsu, Yusi Chen, Guo Wang, Tom Kong, Yineng Zhang, Ziliang Peng and everyone in the channel #community-build-working-group

LMSYS.org sponsors:

NVIDIA, Hyperbolic, Runpod

The First SGLang Online Meetup

SGLang Team

Content

SGLang overview

Recent technique highlights

- CPU overhead hiding
- Fast grammar-guided/JSON decoding
- DeepSeek MLA optimization

Open-source community and roadmap

SGLang Overview

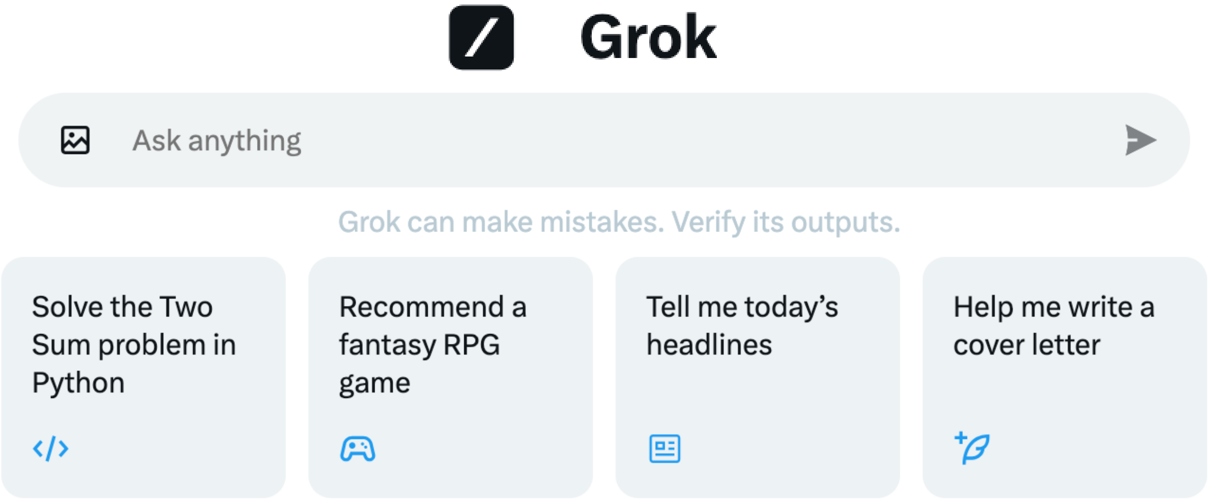
SGLang is a fast serving framework for large language models and vision language models.

What is SGLang?

A **fast inference engine** for LLMs

Comes with its **unique features** for better performance

Serves the **production and research** workloads at multiple institutions



Grok

Ask anything

Grok can make mistakes. Verify its outputs.

Solve the Two Sum problem in Python

Recommend a fantasy RPG game

Tell me today's headlines

Help me write a cover letter

Grok on x.com

SGLang provides leading inference performance

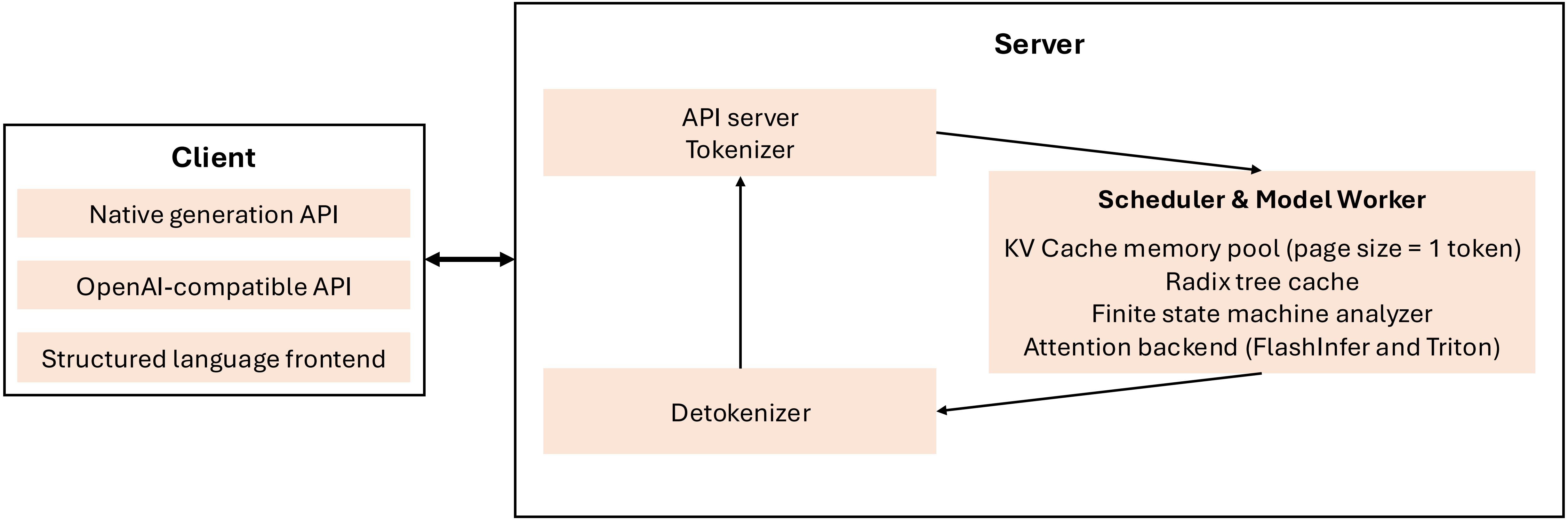
Compared to the other popular inference engines:

v0.1 (Jan. 2024)
5x higher throughput with automatic KV cache reuse
3x faster grammar-based constrained decoding

v0.2 (July 2024)
3x higher throughput with low-overhead CPU runtime

v0.3 (Sept. 2024)
7x faster triton attention backend for custom attention variants (MLA)
1.5x lower latency with torch.compile

SGLang architecture overview



Client

Native generation API

OpenAI-compatible API

Structured language frontend

Server

API server
Tokenizer

Scheduler & Model Worker

KV Cache memory pool (page size = 1 token)
Radix tree cache
Finite state machine analyzer
Attention backend (FlashInfer and Triton)

Detokenizer

Lightweight and customizable code base in Python/PyTorch

Major Techniques

Three techniques covered in this talk

CPU overhead hiding

Step towards fully removing the CPU overhead from the engine

Fast grammar-guided/JSON decoding

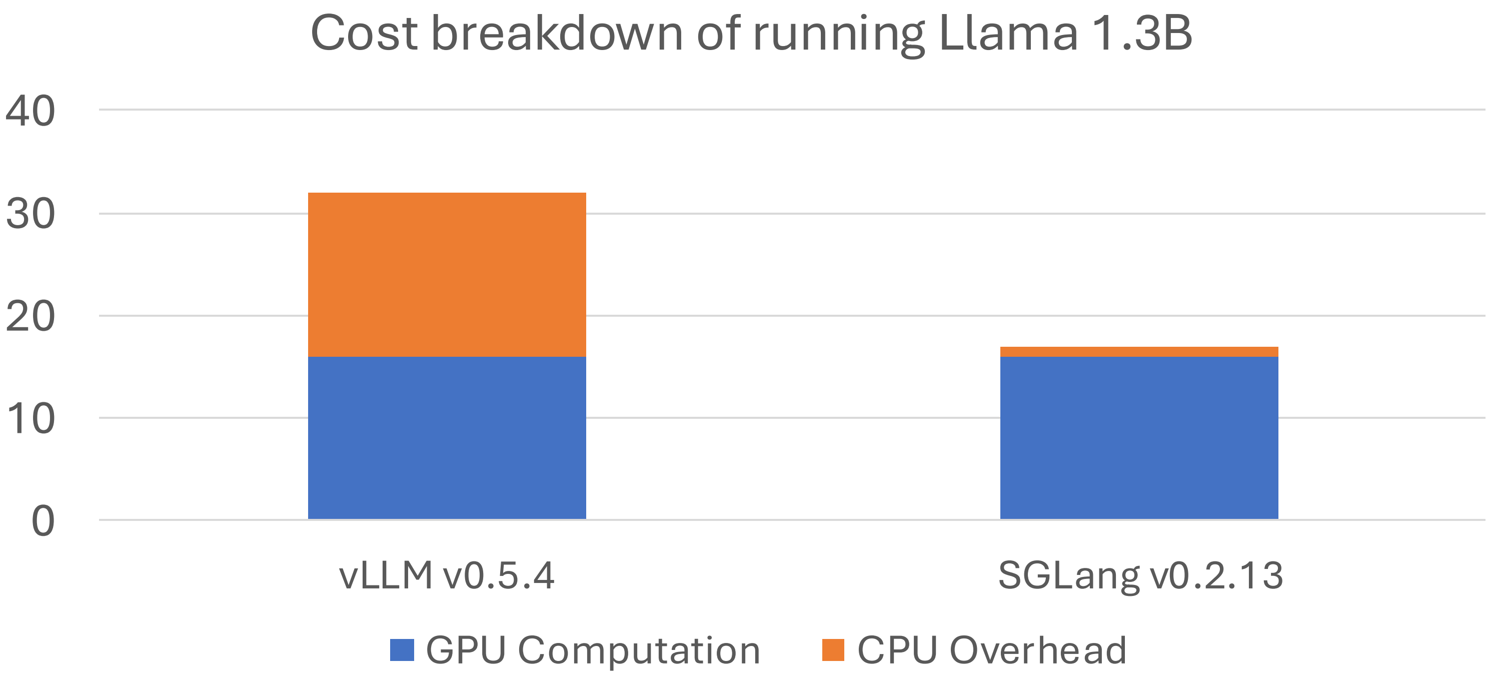
5x faster JSON decoding with the “xgrammar” integration

DeepSeek MLA optimizations

7x higher throughput with optimized kernels for MLA

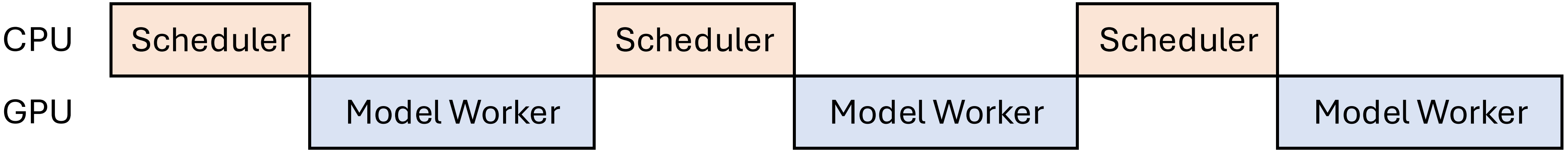
CPU overhead hiding

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



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

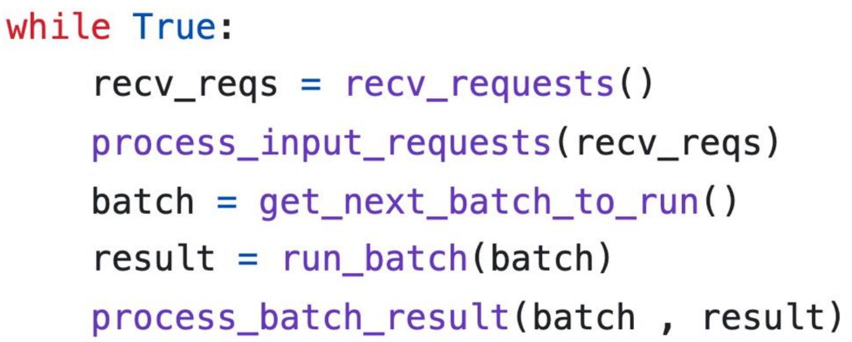
CPU overhead



Jobs of the CPU scheduler

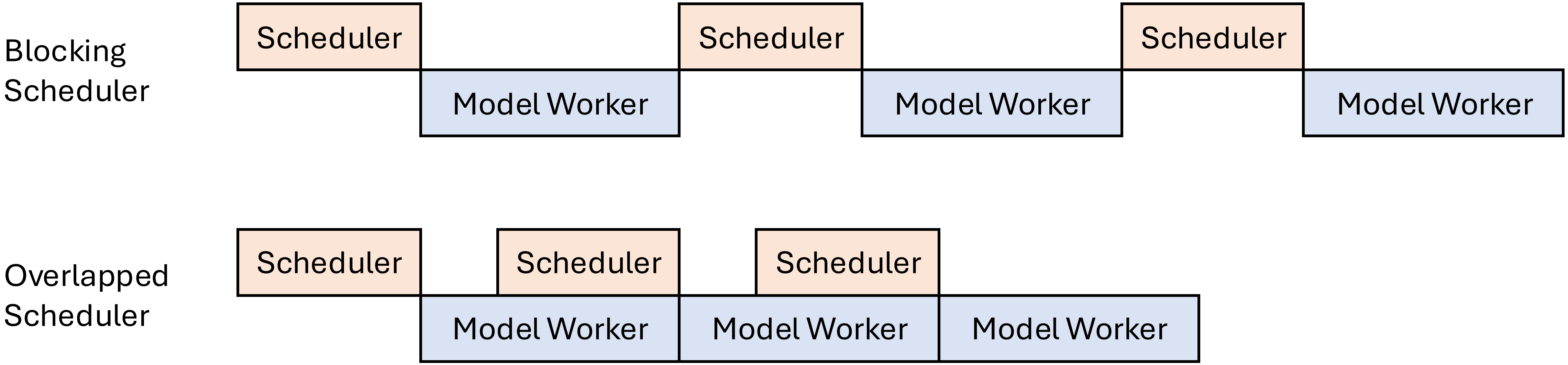
- Receives input messages from the user
- Processes results from the model worker
- Checks the stop conditions
- Runs prefix matching and request reorder
- Allocates memory for the next batch

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



Blocking Scheduler

Scheduler

Model Worker

Scheduler

Model Worker

Scheduler

Model Worker

Overlapped Scheduler

Scheduler

Model Worker

Scheduler

Model Worker

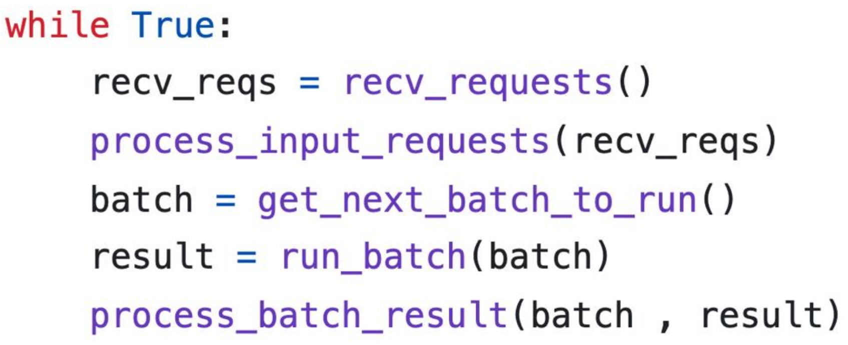
Scheduler

Model Worker

Implementation in the SGLang

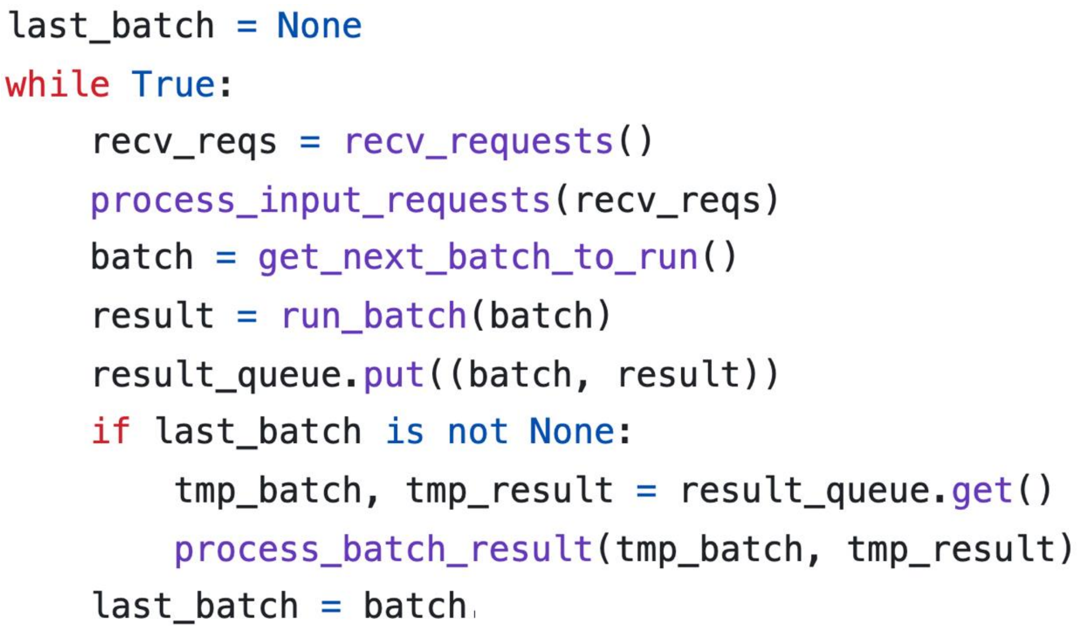
SGLang implements two scheduler loops (#1677, #1687). They share almost all other functions. You can turn on the overlap version with `--enable-overlap`

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
```

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.

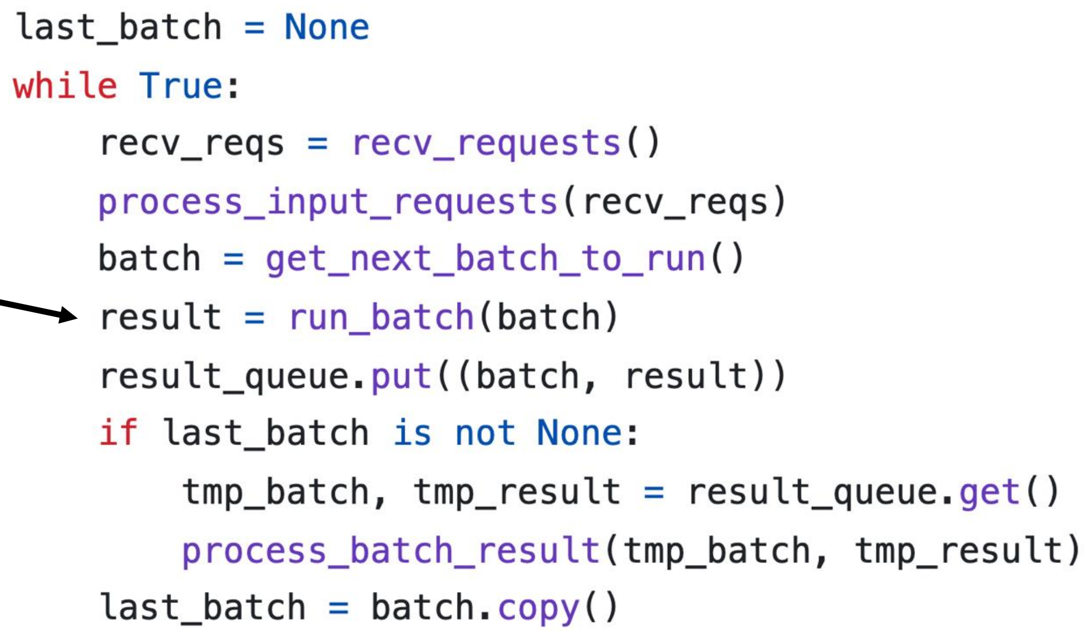
We resolve the dependency by paying the overhead of decoding one more useless token, but this is negelibile.

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



```python
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

Benchmark results

Setup: Llama-3.2-1B. Batch size 256. Input len 4096. Output len 16.

Normal scheduler + many individual HTTP calls	**7.0 s**
↓ Reduce HTTP server overhead	
Normal scheduler + one batched HTTP call	**6.6 s**
↓ Reduce CPU scheduler overhead (mostly radix cache) with the new overlap scheduler	
Overlap scheduler + one batched HTTP call	**6.0 s**
↓ Reduce the remanning overhead	
Goa: Pure GPU computation time	**5.6 s**

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 v0.4.0**

Open-source community and roadmap

Community users and contributors

AMD

xAI

PyTorch

NVIDIA

databricks

RunPod

ByteDance

LinkedIn

deepseek

Hyperbolic

etched

面壁智能 ModelBest

Jam & Tea STUDIOS

Roadmap:
The core team will deliver these features in Q4 2024

Performance optimizations

Sequence parallelism and sparse attention for long context inference

Adaptive speculative decoding for all batch sizes

Disaggregated prefill and decoding

Hierarchical radix cache

Faster grammar parsing libraries

Communication and CPU overhead overlapping

Modular design

Integrate PyTorch-native optimizations

Community building

Bi-weekly online development meeting

Link: https://github.com/sgl-project/sglang/issues/1487

Question & Answer

Github: https://github.com/sgl-project/sglang

Paper (NeurIPS’24) : https://arxiv.org/abs/2312.07104

Welcome to join the slack and bi-weekly dev meeting!