SC26 / TECHNICAL PAPERS

LLMProf

Identifying Performance Bottlenecks in LLM Serving Systems with Top-down Profiling

Tianle Zhong · Mao Lin · Hao Wu · Keren Zhou · Geoffrey Fox

University of Virginia · UC Merced · George Mason University · OpenAI

Why serving needs context

Prefill, decode and changing traffic require different performance questions.

Two phases

Prefill: compute
Decode: bandwidth
Different bottlenecks

Dynamic traffic

Changing batch size
Varying token counts
Mixed model phases

Objectives

Throughput
First-token latency
Per-token latency

Phase and workload context determine the next profiling step.

Paper §I–III; adapted from Paper Story and the online-profiling slides.

What existing profiling views miss

Kernel events, framework traces and service metrics provide different views.

Tool view	What it reveals	Remaining gap
Vendor profiler	Kernel events / metrics	Model and request context
Framework profiler	Operators and call paths	Runtime / finalization cost
End-to-end metric	User-visible symptoms	Root cause and location

LLMProf connects these views to guide the next measurement.

Paper §II–III and Table I; scope of comparisons is the evaluated configurations.

LLMProf design principles

Choose the profiling level, preserve context and collect only relevant metrics.

Start high

Taxonomy guides
the next level
of investigation

Keep context

Workload and model
context down to kernels
Across graph replay

Select metrics

Collect what matters
Refine hot regions
Sample when needed

Shared instrumentation supports NVIDIA and AMD backends.

Paper §IV; adapted from Paper Story and shortIntro.

A diagnosis workflow across three levels

Connect service symptoms to engine behavior and kernel execution.

Service

TTFT · TPOT
Request context

→

Engine

Prefill / decode
CPU vs. GPU time

→

Kernel

Bandwidth · FLOPs
Intra-kernel detail

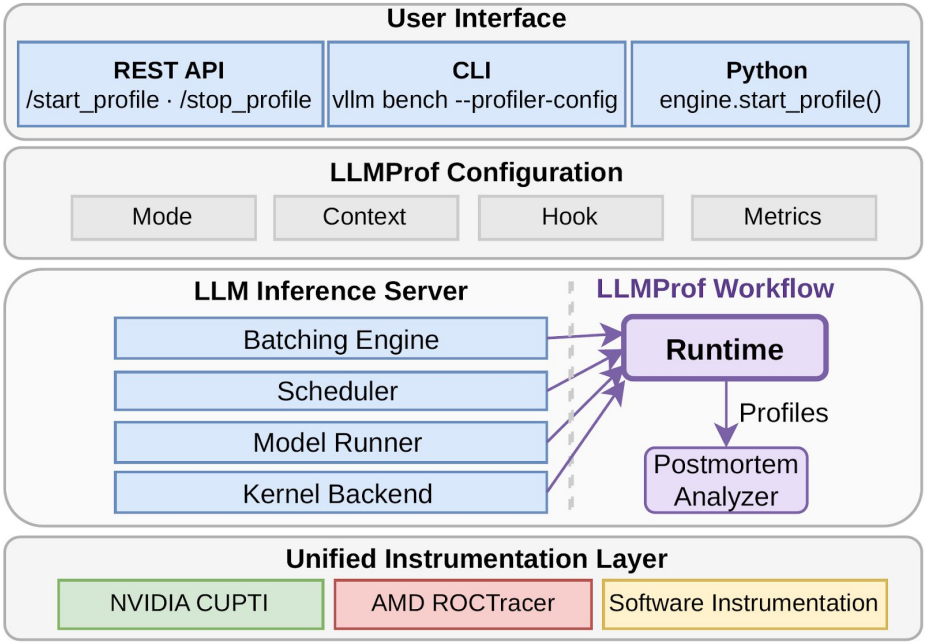
Keep context through capture and replay.

Collect only the metrics needed for the current question.

Paper §IV-B and Table II. TTFT: time to first token; TPOT: time per output token.

LLMProf architecture

Serving controls and model annotations share one profiling runtime.



Annotate once

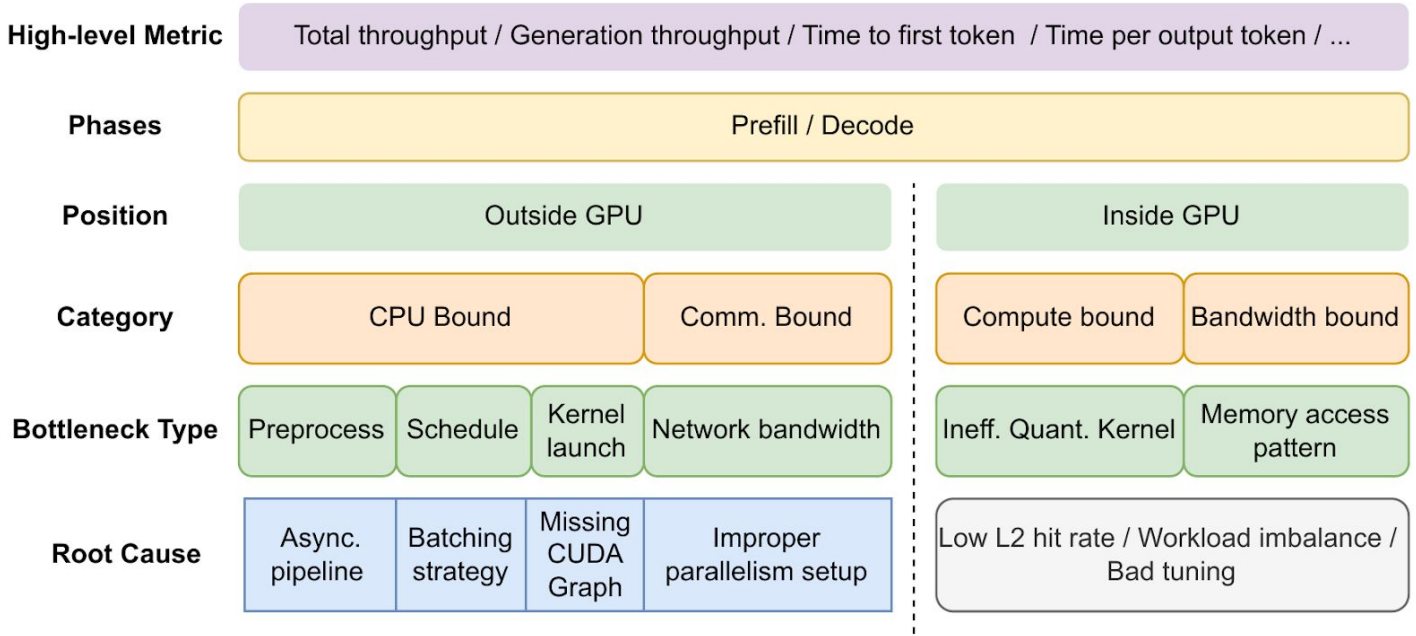
Connect the scheduler, model runner and kernels to one profiling runtime.

Reuse the instrumentation layer across GPU vendors.

Paper Fig. 2 / §IV-A; original architecture diagram.

Performance diagnosis taxonomy

Choose the objective and phase before attributing the bottleneck.



Paper Fig. 1; original diagnosis taxonomy.

Six steps from symptom to bottleneck

Use a coarse pass to find where to look, then refine the measurement.

1 Pick objective

Throughput or latency

2 Find hotspots

Python call paths

3 Add scopes

Tree-style breakdown

4 Add metrics

Shapes, CPU time, bytes

5 Classify cause

Taxonomy + analyzer

6 Inspect kernel

Metadata / sampling

Paper §IV-B, taxonomy-driven workflow.

Workload-aware profiling scopes

Conditional activation isolates the workload; nested scopes retain its meaning.

```
import llmprof as lp
if n_prefill > 1000:
    lp.activate()
    with lp.scope("prefill"):
        model_forward(...)
    lp.deactivate()
```

```
model_forward
  LlamaDecoder
    Attention
    MLP
      gate_up_proj
      down_proj
```

Retain phase, token count and model-layer identity.

Paper §IV-B / Figs. 4–6; pseudocode illustrates the API pattern.

Recovering CUDA Graph context

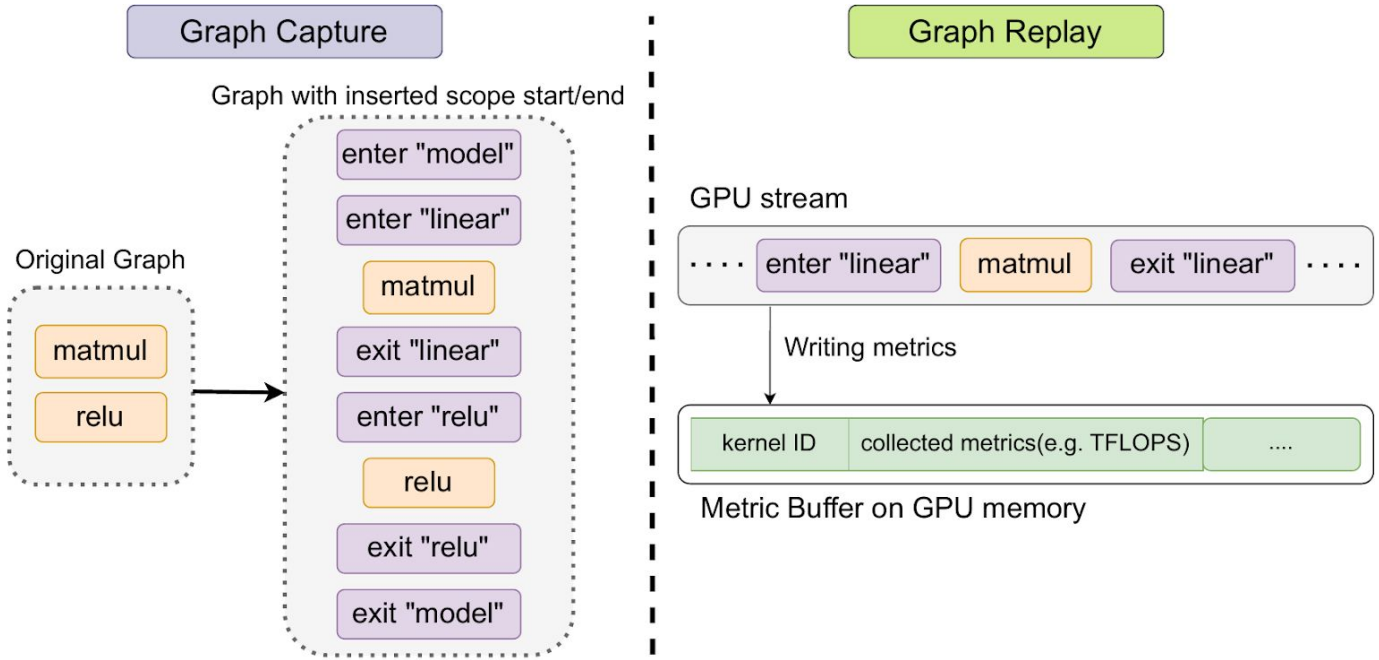
Recovered model scopes explain which operations run inside a replay.

```
[Example profile with CUDA graph profiling]
GPU(ms)   CPU(ms)       | phase/node
0.064     0.081        +- context at replay
0.064     NaN          |  +- <graph-replay>
0.061     NaN          |  |  +- linear
0.061     NaN          |  |  |  +- nvjet_cublas_xxx
0.003     NaN          |  |  +- relu
0.003     NaN          |  |  |  +- elementwise_xxx
-------------------------------------------------------
[Example profile without CUDA graph profiling]
GPU(ms)  CPU(ms)        | phase/node
0.064     0.081        +- context at replay
0.061     NaN          |  +- nvjet_cublas_xxx
0.003     NaN          |  +- elementwise_xxx
```

Paper Fig. 9 / §IV-D; original profile output, times in milliseconds.

Meta kernel profiling

Scope markers record replay metrics in a preallocated GPU buffer.



Paper Fig. 7 / §IV-D; original mechanism diagram.

Postmortem analysis

Hatchet trees support manual inspection and rule-based checks.

Rank

Rank costly scopes within each level

Classify

CPU overhead
Communication time
GPU execution

Check patterns

Find known patterns such as missing CUDA Graph replay

Confirm each diagnostic hypothesis with a new measurement.

Paper §IV-E; automated checks use predefined rules.

Case 1: framework-level bottlenecks

Qwen3-8B · 50 ShareGPT prompts · single NVIDIA H100

Selected GPU work	HF Transformers	vLLM
RMSNorm	259.8 ms	33.0 ms
SiLU + gate	186.2 ms	18.2 ms
Kernel invocations	186,538	52,539

Launch overhead and unfused operations explain the gap.

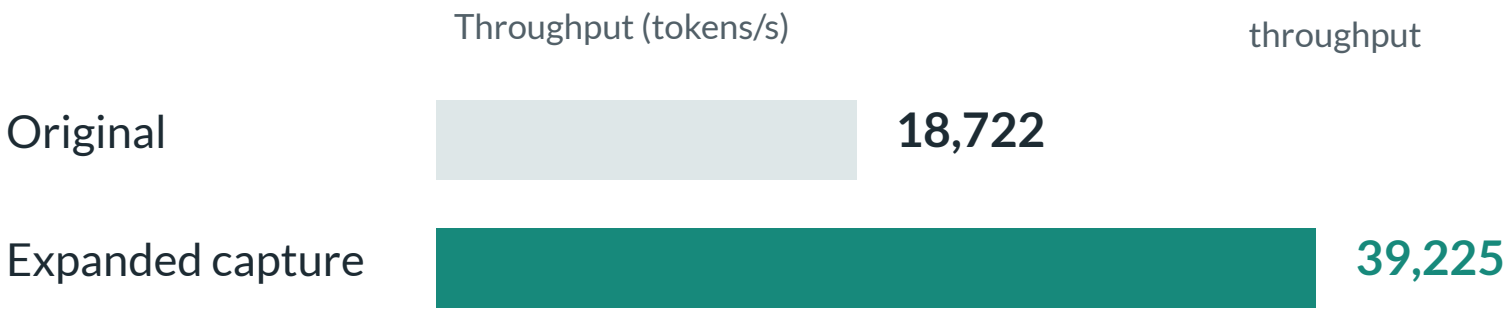
Paper §V-A / Fig. 11. Kernel-time examples are from the single-H100 profile.

Case 2: CUDA Graph coverage

GPT-OSS-20B · vLLM Triton backend · single NVIDIA B200

2.09×

throughput



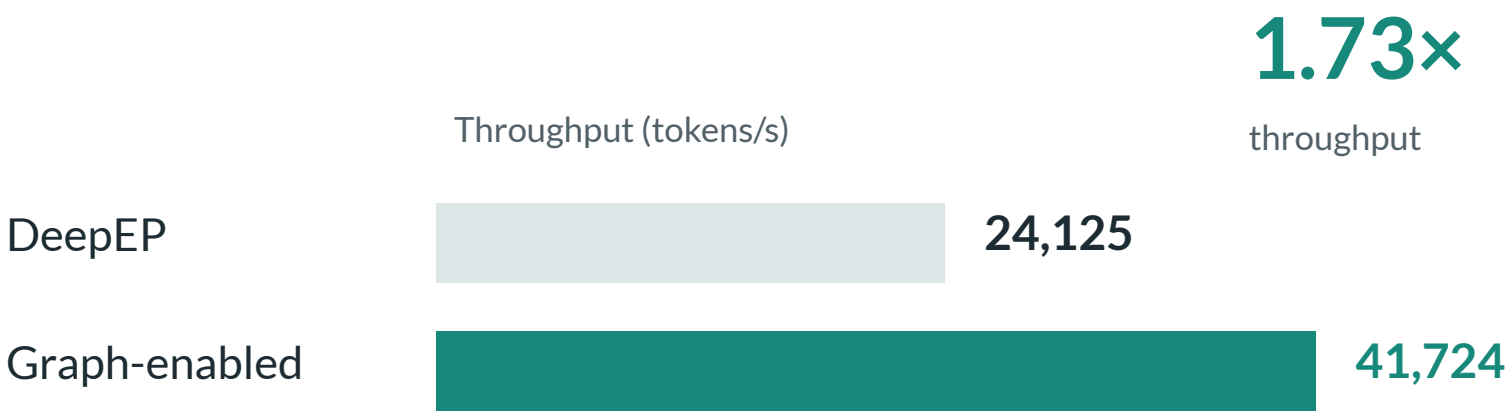
CUDA Graph capture limit: 992 → 8192 tokens

Avoid eager execution for 1,024-token decode batches.

Source: SC26 paper, §V-B / Fig. 12. 2,048 requests; 500 input / 2,000 output tokens each.

Case 3: graphable expert exchange

GPT-OSS-20B · expert parallelism = 2 · two NVIDIA B200 GPUs



P99 TPOT: 126.61 → 90.97 ms

Symmetric-memory all-to-all avoids CPU metadata exchange.

Source: paper §V-B / Fig. 13. EP2 on B200; 1k input / 1k output tokens per request.

Case 4: ineffective swizzling

Intra-kernel profiling reveals where stalled samples accumulate.

```
[Fused MoE | M=64, N=128, K=128, N_experts=64, Topk=2]
stalled samples   | phase/node
19749             +- fused_moe_kernel
2083              |  +- map_pid
8463              |  +- load_data
7034              |  +- compute_loop
1886              |  +- save_data
-------------------------------------------------------
stalled samples   | phase/node
17972             +- fused_moe_kernel_wo_swizzling
1296              |  +- map_pid
7881              |  +- load_data
7123              |  +- compute_loop
1672              |  +- save_data
```

1.05×

kernel speedup

~9% fewer stalled samples

Paper Fig. 14 / §V-C. M=64, N=128, K=128, 64 experts, top-k=2.

Evaluation setup

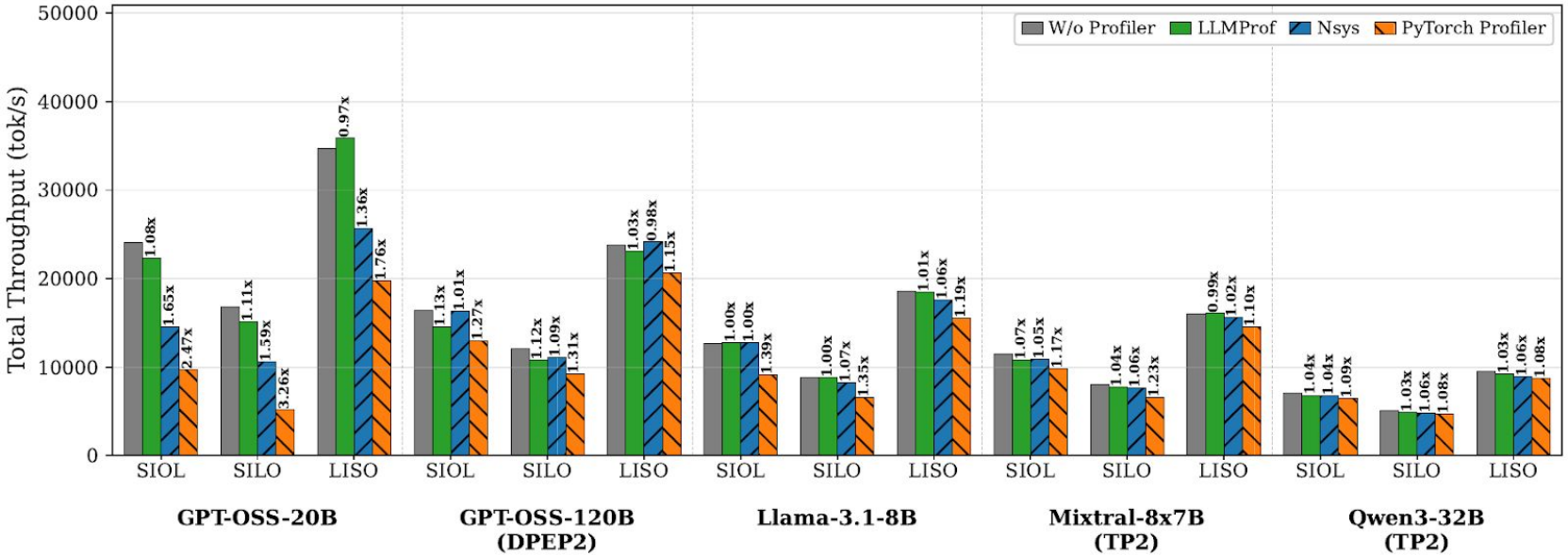
A burst of 2,048 requests across multiple models and input/output lengths.

Dimension	Evaluation setup
NVIDIA	2× H100 80 GB; CUDA 12.8
AMD	1× MI300X; ROCm 7.0
Models	Dense and MoE; single GPU, TP and EP
Input / output	1k/1k, 2k/500 and 500/2k
Integration	vLLM v0.11.2; profile request interval only

Paper §VI-A; NVIDIA and AMD evaluation sets differ.

H100: throughput with profiling

Original per-workload results; higher throughput is better.

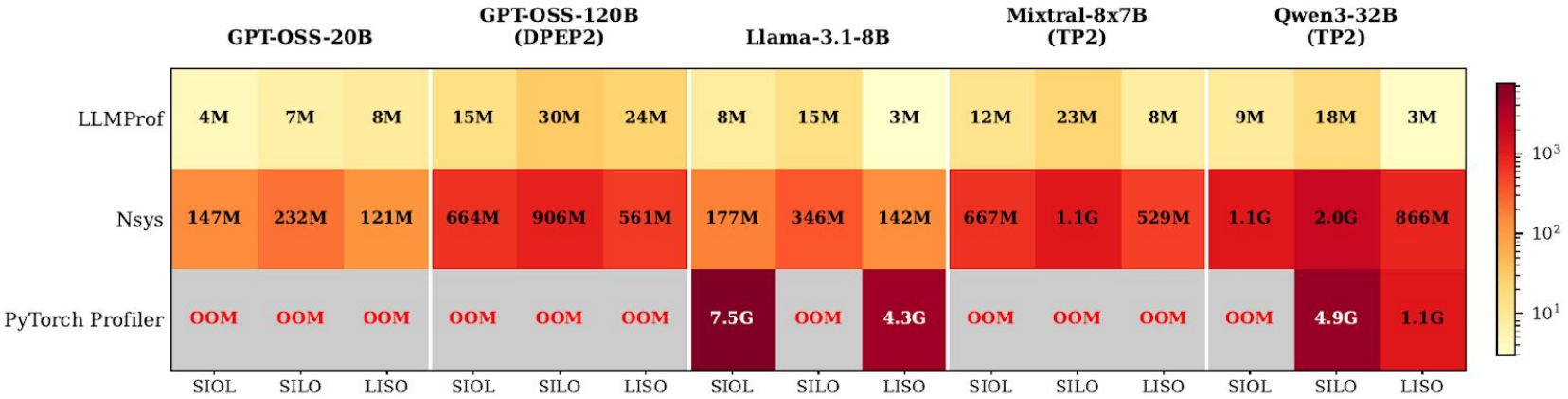


Mean total-throughput slowdown with LLMProf: 1.04×

Paper Fig. 15(a) and Table III. SIOL=1k/1k; SILO=500/2k; LISO=2k/500 input/output.

H100: profile size

Original profile-size heatmap; lower is better.



LLMProf: 3–30 MB | Nsys: up to 2.0 GB

PyTorch often runs out of memory; completed traces can reach GB scale.

Paper Fig. 16(a). Cell values use M/G; color scale is in MB. OOM means out of memory.

Limitations and scope

Instrumentation requirements and the boundaries of the evaluation.

Deployment

Instrument at startup
Cannot attach to an unprepared process

Rule coverage

Predefined patterns
Confirm each cause by measurement

Evidence scope

Hardware-dependent
Workload-dependent
AMD: single GPU

Paper §VII; evaluation boundaries from §VI-A.

Diagnose top down.
Optimize with context.

Connect service, engine and kernel behavior.
Preserve context through graph replay.

Case studies: up to 2.09× end-to-end throughput improvement.

LLMProf | SC26 | Questions?

Reconstructing intra-kernel scopes

Translate annotations into scope boundaries in Triton IR.

1 Pair

Match start and end
Assign scope IDs

2 Validate

Check all CFG paths
Reject invalid scopes

3 Nest

Dominance analysis
Find closest parent

Paper §IV-C. These compiler passes differ from runtime graph context linking.

Mean profiling overhead

Mean slowdown versus unprofiled execution; lower is better.

Platform / tool	Total tput	Gen tput	TTFT	TPOT
H100 / LLMProf	1.04	1.04	1.04	1.06
H100 / Nsys	1.14	1.13	1.18	1.14
H100 / PyTorch	1.46	1.41	1.32	1.43
MI300X / LLMProf	1.00	1.00	1.00	1.00
MI300X / ROCProfV2	1.13	1.05	1.11	1.13
MI300X / PyTorch	1.22	1.22	1.20	1.23

Paper Table III. 1.00× indicates the rounded reported mean, not zero execution cost.

Workloads and platform coverage

Five NVIDIA configurations; two single-GPU configurations on AMD.

Model	Serving mode	H100	MI300X
GPT-OSS-20B	Single GPU	Yes	Yes
Llama-3.1-8B	Single GPU	Yes	Yes
Mixtral-8x7B	TP2	Yes	—
Qwen3-32B	TP2	Yes	—
GPT-OSS-120B	DPEP2	Yes	—

Paper §VI-A. SIOL: 1k/1k; LISO: 2k/500; SILO: 500/2k input/output tokens.

Interpreting kernel measurements

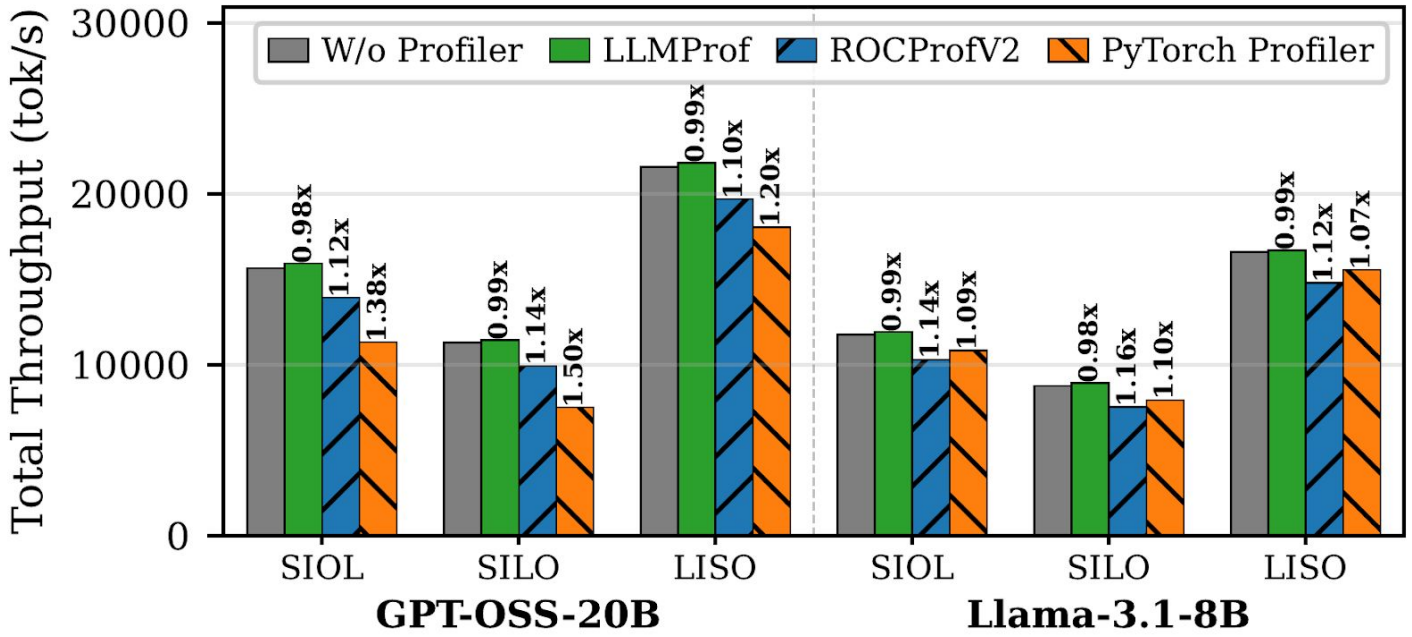
Sampling, timing and speedup support different conclusions.

Measurement	What to infer	What not to infer
PC stalled samples	Likely stall-heavy regions	Exact instruction duration
Intra-kernel cycles	Region time at known clock	Whole-service latency
1.05× kernel speedup	Local optimization benefit	1.05× service throughput

Paper §IV-B–C and §V-C / Fig. 14; values are not end-to-end serving speedups.

AMD MI300X: throughput

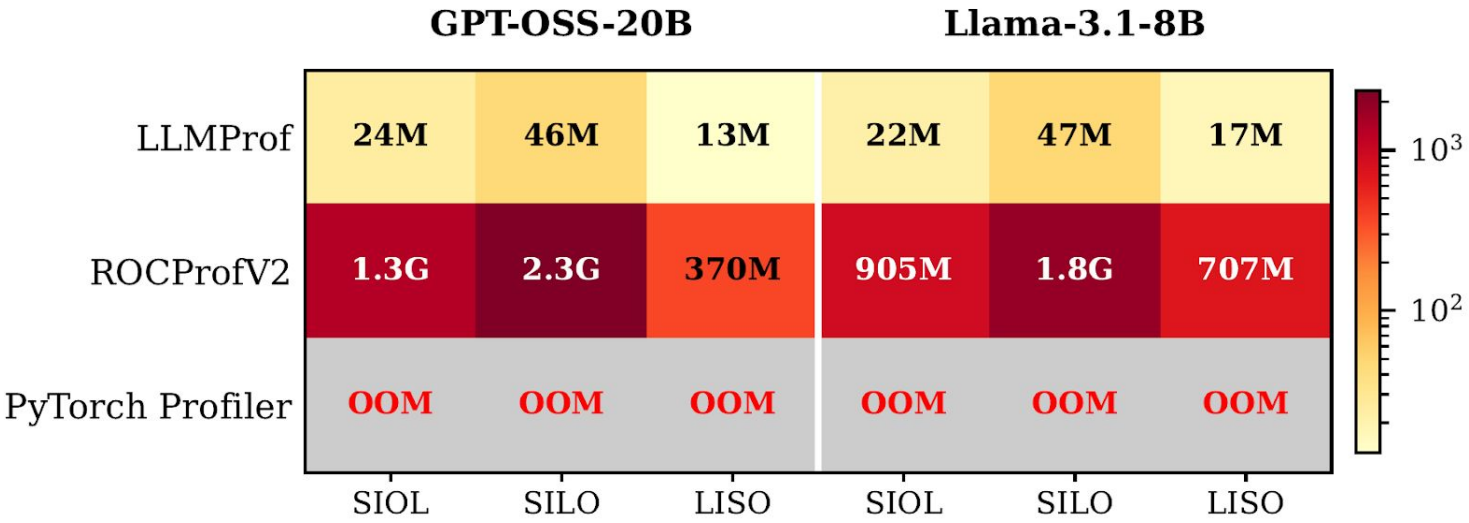
GPT-OSS-20B and Llama-3.1-8B · single GPU · three workloads



Paper Fig. 15(b). SIOL=1k/1k; SILO=500/2k; LISO=2k/500 input/output.

AMD MI300X: profile size

Original measured file sizes and out-of-memory outcomes.



LLMProf: 13–47 MB | ROCProfV2: up to 2.3 GB

Paper Fig. 16(b). Cell values use M/G; color scale is in MB. OOM means out of memory.

CUDA Graph coverage: profile evidence

```
[GPT-OSS-20B | Num Prompt: 2048]
[Max CUDA graph size 992 (default) | Tokens/s: 18722]
GPU(ms)      CPU(ms)       | phase/node
8231.893     1825.911      +- model_forward_decode_1
8231.893     NaN           |  +- <graph-replay>
5340.299     NaN           |  |  +- OAITritonExperts-fused
2717.393     NaN           |  |  |  +- matmul_ogs-w1
2613.319     NaN           |  |  |  +- matmul_ogs-w2
93242.599    163591.589    +- _model_forward_decode_1024
66375.920    121802.423    |  +- OAITritonExperts-fused
40569.116    40745.725     |  |  +- matmul_ogs-w1
24909.552    41462.205     |  |  +- matmul_ogs-w2
-------------------------------------------------
[Max CUDA graph size: 8192 | Tokens/s: 39225]
GPU(ms)      CPU(ms)       | phase/node
91999.172    4115.701      +- model_forward_decode_1024
91999.172    NaN           |  +- <graph-replay>
61702.913    NaN           |  |  +- OAITritonExperts-fused
37996.179    NaN           |  |  |  +- matmul_ogs-w1
22981.731    NaN           |  |  |  +- matmul_ogs-w2
```

2.09×

serving throughput

992 → 8192 tokens

1,024-token decode uses graph replay.

Paper Fig. 12 / §V-B. Each request: 500 input / 2,000 output tokens.

Expert exchange: profile evidence

GPT-OSS-20B · DeepEP · expert parallelism = 2 · NVIDIA B200

```
[GPT-OSS-20B | Num Prompt: 2048 | Tokens/s: 24125]
GPU(ms)   CPU(ms)   kernels | phase/node
141.349   195.29    120     +- _model_forward_prefill_8192
71.149    30.203    48      |  +- OAITritonExperts-fused
47.604    17.255    24      |  |  +- matmul_ogs-w1
23.545    12.170    24      |  |  +- matmul_ogs-w2
35.537    1.9710    48      |  +- deepep_combine
14.857    84.269    24      |  +- deepep_dispatch
-------------------------------------------------------
GPU(ms)   CPU(ms)   kernels   | phase/node
141.710   1068.44    918      +- _model_forward_decode_76
66.562    372.507   726       |  +- OAITritonExperts-fused
39.600    179.823   363       |  |  +- matmul_ogs-w1
26.962    181.052   363       |  |  +- matmul_ogs-w2
13.378    29.765    726       |  +- deepep_combine
15.683    37.906    363       |  +- deepep_dispatch
```

1.73×

serving throughput

Avoid CPU metadata exchange.

Enable graph replay with symmetric-memory all-to-all.

Paper Fig. 13 / §V-B. Each request: 1k input / 1k output tokens.