PR #34645 完整报告

sgl-project/sglang

[AMD][CI] Add GPT-OSS perf benchmarks to the ROCm 7.2 nightly

合并时间：2026-08-17 07:35

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34645

执行摘要

- 一句话：AMD nightly 补 GPT-OSS 性能基准，覆盖 MI30x/MI35x
- 推荐动作：建议精读两个新增测试文件与 generate_simple_markdown_report：一是看如何通过复用 accuracy 的 SERVER_ARGS 保证吞吐量归因正确，二是看 warmup 首行的约定如何与报告函数联动，三是记住多步骤 job 必须清理 GITHUB_STEP_SUMMARY 的教训。设计上最有价值的决策是"先建数据源、后设阈值"——不要在没有历史基线时硬编阈值。后续关注点：积累几周 nightly 数据后为两个 suite 补充吞吐量门禁。

功能与动机

GPT-OSS 是 ROCm 7.2 + Triton 3.7 性能回退报告中的受影响模型之一，但在 AMD 上没有任何吞吐量覆盖，此前只有 DeepSeek-V4 有。PR body 明确说明：'GPT-OSS … has no throughput coverage on AMD — DeepSeek-V4 is the only affected model that does.' 同时 benchmark 暂无 nightly 历史，无法立即设置可辩护的吞吐量阈值，所以本 PR 先建立数据源，为后续回归门禁（如 #34640 的 DeepSeek-V4 gating）铺路。

实现拆解

1. 新增 MI35x 基准测试（test/registered/amd/perf/mi35x/test_gpt_oss_perf_mi35x.py）：通过 register_amd_ci(est_time=3600, suite="nightly-perf-8-gpu-mi35x-gpt-oss", nightly=True) 注册到 AMD nightly。setUpClass 中 batch_sizes = [1, 1, 8, 16, 64]，首元素重复的 1 是 warmup，因为该 benchmark 自己拉起 server，需先支付 JIT 与 autotuning 成本。ENV_VARS 设置 SGLANG_USE_AITER=1 与 SGLANG_USE_AITER_MOE_GU_ITLV=1，原因是 AITER 的 MXFP4 fused MoE 对 gpt-oss 采用分离的 gate/up tile 布局，其他 AITER 调用方默认 interleave，需要显式 opt out。
2. 新增 MI30x 基准测试（test/registered/amd/perf/mi30x/test_gpt_oss_perf_amd.py）：suite 为 nightly-perf-8-gpu-gpt-oss，模型路径换成 lmsys/gpt-oss-*-bf16，ENV_VARS 仅保留 SGLANG_USE_AITER=1（bf16 走 AITER 默认路径，无需 tile 布局开关）。其余结构与 MI35x 版本一致，同样复用 MI30x accuracy 测试的 serving recipe。
3. 提炼共享报告函数（python/sglang/test/nightly_bench_utils.py）：新增 generate_simple_markdown_report(results, default_gpu_config="")，把之前散落在 AMD perf sweep 里的 markdown 渲染逻辑收拢：去掉 H100 定价 cost 列；GPU_CONFIG 缺失时用参数兜底；按"前两条 batch_size 相同"丢弃 warmup 首行；ITL 由

output_throughput 反推（无直接测量值）。两个新测试都复用该函数，保证输出 schema 与现有 AMD perf 报告一致。

4. workflow 追加性能步骤（.github/workflows/nightly-test-amd-rocm720.yml）：在 nightly-accuracy-8-gpu-rocm720 与 nightly-accuracy-8-gpu-mi35x-rocm720 两个 job 内各追加一个 Performance Test step，if: ${{ !cancelled() }}，timeout 120 min，执行 run_suite.py --hw amd --suite nightly-perf-8-gpu-{...} --nightly --timeout-per-file 5400。选择复用 accuracy job 而非新建 job：MI30x 独立 job 拉镜像 + 装依赖约需 49 min，共享后实测成本仅 MI30x 6.5 min / MI35x 3.9 min（约 1.4 GPU-h/night），而独立 job 约 3.4 GPU-h/night。两个 step 均新增 > github_summary.md 清理语句，避免 perf step 把 accuracy 块二次追加到 GITHUB_STEP_SUMMARY（commit e18b280 修复）。
5. 配套验证：性能数据验证 run 31850998638 四步全过；summary 清理修复后 run 31936282742 复验通过。静态校验包含 registry 单文件解析、workflow YAML、重复 job 名、Ruff/Black/isort/codespell。未新增任何精度断言，也无吞吐量阈值。

关键文件：

- test/registered/amd/perf/mi35x/test_gpt_oss_perf_mi35x.py（模块 性能基准；类别 test；类型 test-coverage；符号 TestNightlyGptOssPerformanceMI35x, setUpClass, test_bench_one_batch）：MI35x（gfx950）GPT-OSS MXFP4 吞吐量基准的核心新增文件，定义了 registers、SERVER_ARGS、AITER tile 布局开关与测试主流程，是本 PR 在 gfx950 上的主要数据源。
- test/registered/amd/perf/mi30x/test_gpt_oss_perf_amd.py（模块 性能基准；类别 test；类型 test-coverage；符号 TestNightlyGptOssPerformanceAMD, setUpClass, test_bench_one_batch）：MI30x（gfx942）GPT-OSS bf16 吞吐量基准的新增文件，展示与 MI35x 分裂的 checkpoint 与 AITER 配置差异，是 gfx942 上的主要数据源。
- python/sglang/test/nightly_bench_utils.py（模块 基准工具；类别 test；类型 test-coverage；符号 generate_simple_markdown_report）：新增共享报告函数 generate_simple_markdown_report，统一 AMD perf sweep 的 markdown 输出 schema，并实现 warmup 首行丢弃与 ITL 反推逻辑。
- .github/workflows/nightly-test-amd-rocm720.yml（模块 CI 工作流；类别 infra；类型 infrastructure）：定义性能步骤如何复用 accuracy job，以及 github_summary.md 清理约定；直接决定 nightly 成本与 CI 稳定性。

关键符号：generate_simple_markdown_report, TestNightlyGptOssPerformanceMI35x.setUpClass, TestNightlyGptOssPerformanceMI35x.test_bench_one_batch, TestNightlyGptOssPerformanceAMD.setUpClass, TestNightlyGptOssPerformanceAMD.test_bench_one_batch

关键源码片段

test/registered/amd/perf/mi35x/test_gpt_oss_perf_mi35x.py

MI35x（gfx950）GPT-OSS MXFP4 吞吐量基准的核心新增文件，定义了 registers、SERVER_ARGS、AITER tile 布局开关与测试主流程，是本 PR 在 gfx950 上的主要数据源。

```
"""MI35x nightly performance benchmark for GPT-OSS (8-GPU)."""
```

```
import os
import unittest

from sglang.test.ci.ci_register import register_amd_ci
from sglang.test.nightly_bench_utils import generate_simple_markdown_report
from sglang.test.nightly_utils import NightlyBenchmarkRunner
from sglang.test.test_utils import DEFAULT_URL_FOR_TEST, _parse_int_list_env

register_amd_ci(
    est_time=3600,
    suite="nightly-perf-8-gpu-mi35x-gpt-oss",
    nightly=True,
)

RESULT_DIR = "performance_results_gpt_oss_mi35x"

# MI35x 直接服务 MXFP4 原始 checkpoint；MI30x 只能走 bf16 转换版本。
GPT_OSS_20B_MODEL_PATH = os.environ.get("GPT_OSS_20B_MODEL_PATH", "openai/gpt-oss-20b")
GPT_OSS_120B_MODEL_PATH = os.environ.get("GPT_OSS_120B_MODEL_PATH", "openai/gpt-oss-120b")

# 与 MI35x accuracy 测试完全一致的 serving 参数：吞吐量变化必须归因于
# serving stack，而不是 benchmark recipe 差异。
SERVER_ARGS = [
    "--trust-remote-code",
    "--tp", "8",
    "--attention-backend", "triton",
    "--chunked-prefill-size", "130172",
    "--max-running-requests", "128",
    "--mem-fraction-static", "0.85",
]

# AITER 的 MXFP4 fused MoE 对 gpt-oss 使用分离的 gate/up tile 布局；
# 其他 AITER MXFP4 调用方默认是 interleave，这里显式 opt out。
ENV_VARS = {
    "SGLANG_USE_AITER": "1",
    "SGLANG_USE_AITER_MOE_GU_ITLV": "1",
}


class TestNightlyGptOssPerformanceMI35x(unittest.TestCase):
    """MI35x nightly performance benchmark for the MXFP4 GPT-OSS models."""

    @classmethod
    def setUpClass(cls):
        cls.base_url = DEFAULT_URL_FOR_TEST
        # 首元素重复的 1 是 warmup：该 benchmark 自己拉起 server，
        # 需要先支付 JIT 编译与 autotuning 成本，避免污染首个数据点。
```

```
        cls.batch_sizes = [1, 1, 8, 16, 64]
        cls.input_lens = tuple(_parse_int_list_env("NIGHTLY_INPUT_LENS", "4096"))
        cls.output_lens = tuple(_parse_int_list_env("NIGHTLY_OUTPUT_LENS", "512"))
        cls.models = [GPT_OSS_20B_MODEL_PATH, GPT_OSS_120B_MODEL_PATH]
        cls.runner = NightlyBenchmarkRunner(RESULT_DIR, cls.__name__, cls.base_url)
        cls.runner.setup_result_directory()
        cls.runner.full_report = f"## {cls.__name__}\n"

    def test_bench_one_batch(self):
        """依次 benchmark 两个模型尺寸，任一失败则整体 fail。"""
        failures = []
        env = os.environ.copy()
        env.update(ENV_VARS)
        try:
            for model_path in self.models:
                with self.subTest(model=model_path):
                    results, success, _ = self.runner.run_benchmark_for_model(
                        model_path=model_path,
                        batch_sizes=self.batch_sizes,
                        input_lens=self.input_lens,
                        output_lens=self.output_lens,
                        other_args=SERVER_ARGS,
                        extra_bench_args=["--trust-remote-code"],
                        env=env,
                    )
                    if results:
                        # 与 AMD 既有 perf sweep 相同的输出 schema，直接拼接进报告。
                        self.runner.full_report += (
                            generate_simple_markdown_report(results, "MI35x") + "\n"
                        )
                    if not success:
                        failures.append(f"benchmark failed for {model_path}")
        finally:
            self.runner.write_final_report()
        if failures:
            self.fail("\n".join(failures))
```

test/registered/amd/perf/mi30x/test_gpt_oss_perf_amd.py

MI30x（gfx942）GPT-OSS bf16 吞吐量基准的新增文件，展示与 MI35x 分裂的 checkpoint 与 AITER 配置差异，是 gfx942 上的主要数据源。

```
"""MI30x nightly performance benchmark for GPT-OSS (8-GPU)."""

import os
import unittest

from sglang.test.ci.ci_register import register_amd_ci
from sglang.test.nightly_bench_utils import generate_simple_markdown_report
from sglang.test.nightly_utils import NightlyBenchmarkRunner
```

```
from sglang.test.test_utils import DEFAULT_URL_FOR_TEST, _parse_int_list_env

register_amd_ci(
    est_time=3600,
    suite="nightly-perf-8-gpu-gpt-oss",
    nightly=True,
)

RESULT_DIR = "performance_results_gpt_oss_mi30x"

# MI30x 无法原生服务 MXFP4 checkpoint，只能用社区 bf16 转换版本；
# 模型分裂与既有 accuracy 测试保持一致，保证两个架构各自门禁。
GPT_OSS_20B_MODEL_PATH = os.environ.get(
    "GPT_OSS_20B_BF16_MODEL_PATH", "lmsys/gpt-oss-20b-bf16"
)
GPT_OSS_120B_MODEL_PATH = os.environ.get(
    "GPT_OSS_120B_BF16_MODEL_PATH", "lmsys/gpt-oss-120b-bf16"
)

# 与 MI35x 不同：bf16 走 AITER 默认路径，无需 gate/up tile 布局开关。
ENV_VARS = {"SGLANG_USE_AITER": "1"}


class TestNightlyGptOssPerformanceAMD(unittest.TestCase):
    """MI30x nightly performance benchmark for the bf16 GPT-OSS models."""

    @classmethod
    def setUpClass(cls):
        cls.base_url = DEFAULT_URL_FOR_TEST
        # 首元素重复的 1 是 warmup：自己拉起 server，避免 JIT/autotuning
        # 成本污染首个数据点，与 MI35x 版本行为一致。
        cls.batch_sizes = [1, 1, 8, 16, 64]
        cls.input_lens = tuple(_parse_int_list_env("NIGHTLY_INPUT_LENS", "4096"))
        cls.output_lens = tuple(_parse_int_list_env("NIGHTLY_OUTPUT_LENS", "512"))
        cls.models = [GPT_OSS_20B_MODEL_PATH, GPT_OSS_120B_MODEL_PATH]
        cls.runner = NightlyBenchmarkRunner(RESULT_DIR, cls.__name__, cls.base_url)
        cls.runner.setup_result_directory()
        cls.runner.full_report = f"## {cls.__name__}\n"
```

python/sglang/test/nightly_bench_utils.py

新增共享报告函数 generate_simple_markdown_report，统一 AMD perf sweep 的 markdown 输出 schema，并实现 warmup 首行丢弃与 ITL 反推逻辑。

```
def generate_simple_markdown_report(
    results: List[BenchmarkResult], default_gpu_config: str = ""
) -> str:
    """生成不带 H100 定价 cost 列的 markdown 报告，并丢弃 warmup 首行。

    warmup 由调用方通过重复第一个 batch size 表达：当前两条结果
```

```
    batch_size 相同且结果多于一行时，去掉第一条（JIT/autotuning 预热）。
    """
    model_header = results[0].model_path
    if results[0].run_name and results[0].run_name != "default":
        model_header += f" ({results[0].run_name})"

    # 优先取环境变量 GPU_CONFIG，缺失时退回调用方传入的架构名。
    gpu_config = os.getenv("GPU_CONFIG", default_gpu_config)
    if gpu_config:
        model_header += f" [{gpu_config}]"

    summary = f"### {model_header}\n"
    summary += (
        "| batch size | input len | latency (s) | "
        "input throughput (tok/s) | output throughput (tok/s) | ITL (ms) |\n"
    )
    summary += (
        "| ---------- | --------- | ----------- | "
        "------------------------ | ------------------------- | -------- |\n"
    )

    report_results = (
        results[1:]
        if len(results) > 1 and results[0].batch_size == results[1].batch_size
        else results
    )

    for result in report_results:
        # ITL 没有直接测量值，由 output_throughput 反推：
        # 1 / ( 吞吐 / batch_size) * 1000。
        itl = (
            1 / (result.output_throughput / result.batch_size) * 1000
            if result.output_throughput > 0
            else 0
        )
        summary += (
            f"| {result.batch_size} | {result.input_len} | {result.latency:.2f} | "
            f"{result.input_throughput:.2f} | {result.output_throughput:.2f} | "
            f"{itl:.2f} |\n"
        )
    return summary
```

评论区精华

review 的核心交锋集中在 PR body 的 Speed Test 表格上。HaiShaw 两次追问性能差异：第一次问两套架构表为何差距明显，michaelzhang-ai 答为独立基线（MI35x gfx950 + MXFP4 vs MI30x gfx942 + bf16，精度分裂由硬件能力决定），并用权重足迹佐证 120b 在 MI30x 上流量多 2.38x 而吞吐比 2.41x 吻合；第二次追问同一表内两组数据差异，michaelzhang-ai

承认先前答错，真实原因是 20b 与 120b 两个模型的独立扫描被表格展示成一组，同时暴露了一个真 bug——新增 perf step 未清理 github_summary.md，导致 accuracy 块被重复追加到 GITHUB_STEP_SUMMARY，已在 commit e18b280 修复并复验。

- 两套架构性能数据差异的解释 (question): 确认为独立基线而非同工作负载重复；跨架构吞吐比与权重流量比吻合，数据可信。
- 同一表格内两组数据差异与 GITHUB_STEP_SUMMARY 重复渲染 (correctness): 展示层是 PR 描述 bug，测量本身正确；CI 渲染 bug 在 commit e18b280 修复（两个 step 均加 > github_summary.md），复验 run 31936282742 通过。

风险与影响

- 风险：技术风险集中在 CI 基础设施层，不涉及运行时 serving 代码。（1）nightly 稳定性：性能步骤失败会让所在 job 变红，if: !cancelled() 避免 accuracy 失败后空跑，但 benchmark 错误（如 Hugging Face 拉取失败、超时）仍可能污染 nightly 红绿状态。（2）无自动回归门禁：当前只报告不下结论，ROCm 7.2 slowdown 的自动捕获需要后续基于这批基线设置阈值。（3）warmup 检测是隐含契约：generate_simple_markdown_report 以“前两条 batch_size 相同”判断 warmup，后续新调用方若不遵守“重复首元素”约定，行为可能不符合预期（函数 docstring 已说明）。（4）外部 checkpoint 依赖：lmsys/gpt-oss-*-bf16 为社区转换版本，上游变动会悄然改变数据前提；MI35x 的 MXFP4 路径依赖 AITER tile 布局开关，AITER 升级可能改变 SGLANG_USE_AITER_MOE_GU_ITLV 语义。
- 影响：影响范围集中在 AMD ROCm 7.2 nightly 测试基础设施：MI30x/MI35x 各自新增 20b/120b 两组吞吐量扫描，每次 4 个 batch size，新增成本约 1.4 GPU-h/night（复用 accuracy job 后比独立 job 节省约 2 GPU-h/night）。不触碰任何运行时 serving 代码，对用户请求路径零影响。对团队而言，这为 AMD 性能回归守门提供了第一批可量化的基线数据，与 #34640 的 DeepSeek-V4 gating 一起构成 ROCm 7.2 slowdown 的追踪闭环。
- 风险标记：CI 稳定性影响：perf 步骤失败会使 job 变红，无吞吐量阈值，暂不能自动捕获性能回归，外部 checkpoint 依赖（Hugging Face 拉取），warmup 丢弃逻辑依赖调用方约定

关联脉络

- PR #34640 DeepSeek-V4 gating（标题未提供，按 PR body 引用）: PR body 明确说明 'DeepSeek-V4 gating is separate in #34640'；DeepSeek-V4 是此前唯一有 AMD 吞吐量覆盖的受影响模型，与本 PR 同属 ROCm 7.2 + Triton 3.7 slowdown 追踪的姊妹 PR。