

PR #38908 完整报告

sgl-project/sglang

Fix gpt-oss RunAI streamer weight ownership

合并时间: 2026-09-12 20:21

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38908>

执行摘要

- 一句话: 修复 gpt-oss RunAI 流式加载权重被缓冲区覆写导致非有限 logits
- 推荐动作: 值得精读, 篇幅小 (2 文件、+126/-12) 但信息密度高, 是「零拷贝视图所有权」这类缺陷的典型教材。阅读时重点看三处设计决策: 其一, 为何专家权重用生成器惰性消费而不是 clone 全量——因为专家权重是 checkpoint 主体, 且 `_load_mxfp4_experts_weights` 本身即到即拷, 惰性化零成本地消除了风险; 其二, 为何保留副本必须落 host 而非 device——这正是 Codex P1 抓到的显存峰值问题, 也是最值得借鉴的 review 结论; 其三, 测试如何在没有模型和 GPU 的前提下构造反例——`object.__new__` 绕过 `__init__`、两个 loader 半被替换为记录器, 用 `[3, 3, 3]` vs `[1, 2, 3]` 精确刻画「跑到流前面」的语义。后续跟进建议: 把 `_own_if_runai_streamed` 收敛进共享 `runai_utils` (视 #37371 落地情况), 并单独评估 `_canonicalize_weights` 中 `dict(weights_in)` 的暴露面与修复代价。

功能与动机

RunAI 流式器在所有张量之间复用同一块 staging buffer, model loader 只要持有跨 yield 的零拷贝视图, 就可能在读取前被覆写。PR body 明确指出 `_load_weights_mxfp4` 的旧实现 "Nothing is read until the iterator is exhausted, so every tensor is retained for the length of the stream", 结果是 "The model then loads with no error onto weights that are partly garbage, and every request comes back with non-finite logits"。作者把它归为同一类缺陷, 此前已有 DeepSeek 修复 (#32913)、GLM5 修复 (#37370) 和仍在推进的 NVFP4 修复 (#37371), 而 "gpt-oss has neither a guard nor an issue"。体感证据来自 PR body 的对照表: `openai/gpt-oss-20b` 在 `--load-format runai_streamer` 且 `RUNAI_STREAMER_MEMORY_LIMIT=1158266880` 时输出全为 ! (贪婪采样落在非有限 logits 上), 加补丁后恢复为正常的 "Paris.", 与 `safetensors` 基线一致。

实现拆解

变更入口是 `python/sglang/srt/models/gpt_oss.py` 的 `GptOssForCausalLM.load_weights`, 它按 `quant_config.get_name()` 分派到 `_load_weights_mxfp4` (`mxfp4`)、`load_gptoss_weight_quark` (`quark`) 或 `_load_normal_weights`; 本 PR 只改 `mxfp4` 分支和新增的模块级 helper, 其余分派路径不变。

1. 定位问题: 把 `weights` 迭代器一次性抽干 (`_load_weights_mxfp4`)。旧实现先 `for name, weight in weights` 把整条 checkpoint 分流进 `mxfp4_weights` / `normal_weights`

- 两个 list，再调用 `_load_mxfp4_experts_weights(mxfp4_weights)`。`weights` 来自 `runai_safetensors_weights_iterator`，`yield` 的是 staging buffer 的零拷贝视图；迭代器耗尽后列表里的视图已被后续张量复用覆盖，于是权重被「静默加载成垃圾」。
2. 专家权重：列表改为惰性生成器（内层函数 `experts`）。新的内层生成器对 `.experts` 且 `mxfp4` 的条目直接 `yield name, weight`，并把生成器本身传入 `self._load_mxfp4_experts_weights(experts(weights))`。`_load_mxfp4_experts_weights` 在循环内即时把每个张量拷进参数，因此专家张量的生命周期不跨越下一次 `yield`，天然不受缓冲区复用影响；专家权重又是 checkpoint 的主体，惰性化也顺带避免整份专家权重常驻内存。
 3. 非专家权重：显式取得所有权（`_own_if_runai_streamed`）。非专家权重必须等 `_load_mxfp4_experts_weights` 返回后才交给 `_load_normal_weights`，因此改为 `normal_weights.append((name, _own_if_runai_streamed(weight)))`。新增的模块级 helper 仅在 `getattr(tensor, RUNAI_STREAMER_TENSOR_ATTR, False)` 为真时返回副本，否则原样返回，非 RunAI 路径零额外开销；文件顶部 `import` 相应扩展为 `(RUNAI_STREAMER_TENSOR_ATTR, default_weight_loader)`。
 4. review 驱动的修正：副本从 device 移到 host（commit 22e35dc）。首版 helper 名为 `_clone_if_runai_streamed_tensor`，直接在 device 上 `clone()`。Codex 以 P1 指出：对象存储 URI 默认启用分布式流式，产出的是 device 张量，逐一 `clone` 产生的独立分配会一直存活到 checkpoint 读完，抬高数 GiB 峰值显存且不计入 `RUNAI_STREAMER_MEMORY_LIMIT`，可能让单卡 gpt-oss 在加载阶段 OOM。作者据此改为 `tensor.detach().to("cpu", copy=True)`，并保留表达「取得所有权」语义的函数名 `_own_if_runai_streamed`；单测同步增加 `held.device.type == "cpu"` 断言。
 5. 测试配套（新增 `test/registered/unit/models/test_gpt_oss_runai_ownership.py`）。用 `register_cpu_ci(est_time=5, suite="base-a-test-cpu")` 注册进 CPU CI；`_streamed()` 构造带 `RUNAI_STREAMER_TENSOR_ATTR` 标记的 `torch.full((4,), value)` 模拟缓冲区视图，`_Mxfp4QuantConfig.get_name()` 返回 "mxfp4"，`_model()` 用 `object.__new__(GptOssForCausalLM)` 绕过 `__init__`，把 `_load_mxfp4_experts_weights` / `_load_normal_weights` 替换成记录器，全程无 GPU、无真实权重。`test_expert_weights_are_consumed_as_they_arrive` 断言产出与消费严格交替（未打补丁时为 [3, 3, 3]，打补丁后 [1, 2, 3]）；`test_retained_weights_are_copied_out_of_the_buffer` 断言保留张量与原始视图非同一对象、`data_ptr` 不同、落在 CPU、数值一致，而未标记的 `lm_head.weight` 仍原样透传（`assertIs`）。

维度	变更前	变更后
专家权重生命周期	攒完再说，跨整条流	生成器即到即拷
非专家权重内存	复用 streamer 缓冲区视图	RunAI 标记张量持有 host 副本
峰值显存	基线	与基线持平（避免 device clone）
host 内存	基线	增加数 GiB（非专家权重副本）
回归保护	无	2 个 hermetic CPU 单测

关键文件：

- `python/sglang/srt/models/gpt_oss.py`（模块 权重加载；类别 `source`；类型 `core-logic`；符号 `GptOssForCausalLM.load_weights`, `_load_weights_mxfp4`, `experts`, `normal_weights`）：唯一的源码主文件：把 `_load_weights_mxfp4` 的『先攒后读』改成生成器惰性消费，并新增 `_own_if_runai_streamed` 为必须保留的非专家权重取得 `host` 副本；同时扩展 `RUNAI_STREAMER_TENSOR_ATTR` 的 `import`。加载正确性与显存 / 内存代价的权衡全部落在这个文件里。
- `test/registered/unit/models/test_gpt_oss_runai_ownership.py`（模块 单元测试；类别 `test`；类型 `test-coverage`；符号 `_streamed`, `_Mxfp4QuantConfig`, `TestGptOssRunaiOwnership`, `_model`）：新增的 `hermetic` 回归测试，用两个纯 Python 检查同时锁定本次修复的两个关键不变量：专家权重即到即用、被保留的张量脱离流式器缓冲区且落在 CPU。两个用例在未打补丁的 loader 上会失败，构成真实回归保护，并已注册进 `base-a-test-cpu CPU CI`。

关键符号：`_load_weights_mxfp4`, `experts`, `_own_if_runai_streamed`,
`_load_mxfp4_experts_weights`, `_load_normal_weights`,
`test_expert_weights_are_consumed_as_they_arrive`,
`test_retained_weights_are_copied_out_of_the_buffer`,
`TestGptOssRunaiOwnership._model`

关键源码片段

`python/sglang/srt/models/gpt_oss.py`

唯一的源码主文件：把 `_load_weights_mxfp4` 的『先攒后读』改成生成器惰性消费，并新增 `_own_if_runai_streamed` 为必须保留的非专家权重取得 `host` 副本；同时扩展 `RUNAI_STREAMER_TENSOR_ATTR` 的 `import`。加载正确性与显存 / 内存代价的权衡全部落在这个文件里。

```
# 文件顶部 import 扩展：从 weight_utils 一并引入 RunAI 流式器标记
from sglang.srt.model_loader.weight_utils import (
    RUNAI_STREAMER_TENSOR_ATTR,
    default_weight_loader,
)
```

```
def _load_weights_mxfp4(self, weights, is_nextn, weight_name_mapping):
    normal_weights = []
```

```
def experts(weights):
    # RunAI 流式器在所有张量之间复用同一块 staging buffer，因此某个张量若在
    # 其后续张量到达之后才被读取，读回的可能已是垃圾数据。专家权重在被 yield
    # 的当下就会拷入参数，可以安全地惰性消费；其余权重会一直留到流结束，
    # 必须自持内存。
    for name, weight in weights:
        if (
```

```

        ".experts" in name
        and self.quant_config is not None
        and self.quant_config.get_name() == "mxfp4"
    ):
        yield name, weight
    else:
        normal_weights.append((name, _own_if_runai_streamed(weight)))

# 生成器被完整消费：专家权重即到即拷，不再在列表里积压一整条流
mxfp4_loaded_params = self._load_mxfp4_experts_weights(experts(weights))
self._load_normal_weights(
    normal_weights,
    is_nextn=is_nextn,
    weight_name_mapping=weight_name_mapping,
    other_loaded_param_names=mxfp4_loaded_params,
)

```

```
def _own_if_runai_streamed(tensor: torch.Tensor) -> torch.Tensor:
```

```
    """取一份流式器无法覆写的副本。
```

副本落在 host 上：分布式流式读取会产出 device 张量，而这些张量要一直保留到整个 checkpoint 读完；若就地 clone，峰值显存会多出数 GiB，且不计入流式器配置的内存上限（RUNAI_STREAMER_MEMORY_LIMIT）。

```
    """
```

```
    if getattr(tensor, RUNAI_STREAMER_TENSOR_ATTR, False):
```

```
        # copy=True 强制分配新存储，detach 切断与原视图的计算图关联
```

```
        return tensor.detach().to("cpu", copy=True)
```

```
    # 未打 RunAI 标记（如本地 safetensors）时原样透传，零额外开销
```

```
    return tensor
```

test/registered/unit/models/test_gpt_oss_runai_ownership.py

新增的 hermetic 回归测试，用两个纯 Python 检查同时锁定本次修复的两个关键不变量：专家权重即到即用、被保留的张量脱离流式器缓冲区且落在 CPU。两个用例在未打补丁的 loader 上会失败，构成真实回归保护，并已注册进 [base-a-test-cpu](#) CPU CI。

```
register_cpu_ci(est_time=5, suite="base-a-test-cpu")
```

```
def _streamed(value: float) -> torch.Tensor:
```

```
    """构造一个带 RunAI 标记的张量，模拟 staging buffer 的零拷贝视图。"""
```

```
    tensor = torch.full((4,), value)
```

```
    setattr(tensor, RUNAI_STREAMER_TENSOR_ATTR, True)
```

```
    return tensor
```

```
class _Mxfp4QuantConfig:
```

```
    def get_name(self) -> str:
```

```
        # 仅用于把 load_weights 分派到 mxfp4 分支
```

```
return "mxfp4"
```

```
class TestGptOssRunaiOwnership(CustomTestCase):
    def _model(self, on_experts, on_normal):
        # 不跑 __init__, 用 object.__new__ 造出模型壳子, 再把 loader 的两半
        # 换成记录器, 测试无需 GPU、无需真实权重
        model = object.__new__(GptOssForCausalLM)
        model.quant_config = _Mxfp4QuantConfig()
        model._load_mxfp4_experts_weights = on_experts
        model._load_normal_weights = on_normal
        return model

    def test_expert_weights_are_consumed_as_they_arrive(self):
        produced = []

        def stream():
            for i in range(3):
                produced.append(i)
                yield f"model.layers.{i}.mlp.experts.gate_up_proj_blocks", _streamed(i)

        produced_when_seen = []

        def on_experts(weights):
            for _name, _weight in weights: # 直到这里才真正向流拉取张量
                produced_when_seen.append(len(produced))
            return set()

        model = self._model(on_experts, lambda *a, **k: None)
        model._load_weights_mxfp4(stream(), is_nextn=False, weight_name_mapping=None)

        # 一产一消: loader 不会跑到流前面、把先前视图晾在缓冲区上
        # 未打补丁时这里是 [3, 3, 3], 即三个张量先全部产出再统一消费
        self.assertEqual(produced_when_seen, [1, 2, 3])

    def test_retained_weights_are_copied_out_of_the_buffer(self):
        streamed = _streamed(1.0) # 带标记的「缓冲区视图」
        plain = torch.full((4,), 2.0) # 未标记, 应该原样透传
        kept = {}

        def on_normal(weights, **kwargs):
            kept.update({name: tensor for name, tensor in weights})

        model = self._model(lambda weights: {n for n, _ in weights}, on_normal)
        model._load_weights_mxfp4(
            iter([("model.embed_tokens.weight", streamed), ("lm_head.weight", plain)]),
            is_nextn=False,
            weight_name_mapping=None,
        )
```

```

held = kept["model.embed_tokens.weight"]
self.assertIsNot(held, streamed) # 不再是同一个对象
self.assertNotEqual(held.data_ptr(), streamed.data_ptr()) # 不再共享存储
# 要留到流结束，所以应落在 host，而不是流式器内存上限管不到的 device 内存
self.assertEqual(held.device.type, "cpu")
torch.testing.assert_close(held, streamed)

# 未标记的张量不做无谓复制
self.assertIs(kept["lm_head.weight"], plain)

```

评论区精华

1. Codex P1: device 端 clone 会吃掉显存上限（性能，已解决）。Codex 在 [python/sglang/srt/models/gpt_oss.py](#) 的 diff 上指出: "For RunAI loads from object-storage URIs on CUDA, distributed streaming is enabled by default and [runai_safetensors_weights_iterator](#) therefore yields device tensors. Cloning every non-expert tensor here creates independent CUDA allocations that all remain referenced by [normal_weights](#) until the entire checkpoint has streamed ... This can make the single-GPU GPT-OSS configurations OOM while loading." 这是本 PR 最有价值的一击：首版修复把正确性问题换成了资源问题。
2. 作者回应并修正 (commit 22e35dc) : "[_is_distributed](#) defaults to [is_object_storage_path](#), so exactly the loads this patch targets are the ones yielding device tensors, and cloning in place would have traded a correctness bug for an OOM ... Peak GPU usage is unchanged from before this PR; the cost is a few GiB of host memory and one D2H copy of the non-expert weights." 单测同步断言保留副本落在 CPU。
3. 维护者态度: [ispobock](#) 直接 APPROVED 并执行合并; [ekzhang](#) 只问了 "did your robot make a stray PR?", [knlb-tml](#) 澄清 "sam asked me to make this upstream explicitly -- just cleaning it up based on chatgpt feedback", 说明这是有意上游化的修复而非误提。
4. 未在 review 中展开但作者主动记录的遗留项（仍待跟进）：一是 [_canonicalize_weights](#) 开头的 [dict\(weights_in\)](#) 让非 mxfp4 路径暴露面相同，作者选择不动；二是本地 helper 与 #37371 计划引入的共享 [sglang.srt.model_loader.runai_utils](#) 存在重复，作者建议后续合并，避免 "becoming a fourth copy"。
 - 保留的非专家权重不应在 device 上 clone（会抬高峰值显存）（performance）: 作者认可并在 commit 22e35dc 改为 `tensor.detach().to("cpu", copy=True)`，helper 更名为 `_own_if_runai_streamed`，单测补充 `held.device.type == "cpu"` 断言；峰值 GPU 显存与修复前持平，代价是数 GiB host 内存与一次 D2H 拷贝。
 - 非 mxfp4 路径 `_canonicalize_weights` 的同类暴露面 (design): 本次刻意不修，作为已知遗留暴露面保留；review 中未有人反对该取舍，但也没有明确的上游负责人或 follow-up issue。
 - ownership helper 与共享 `runai_utils` 的重复实现 (design): 暂以本文件内的 `_own_if_runai_streamed` 落地，等待 #37371 的共享模块后收敛；存在版本漂移与行为不一致的长期风险。

- 该 PR 是否为机器人误提 (stray PR) (question): 确认为有意上游化的修复, 并按 ChatGpt review 反馈完成清理, 无流程争议。

风险与影响

- 风险:

1. 同类暴露面未清除 (正确性, 中): `_canonicalize_weights` 仍以 `dict(weights_in)` 开头, 会一次性持有整份迭代器的视图, 作者在 PR Notes 中明确 "the non-mx_fp4 path has the same exposure" 并刻意不修, 理由是等价修法要把整份 checkpoint 拷进 host 内存, 代价大于收益。因此 gpt-oss 走非 mxfp4 / 非 `_load_weights_mxfp4` 的 RunAI 加载路径仍可能复现同类静默损坏。
1. 生成器带来的隐式耦合 (正确性, 中): `normal_weights` 的完整性现在取决于 `_load_mxfp4_experts_weights` 是否完整耗尽 `experts(weights)`。若该函数将来出现提前 `break/return` 或跳过某些层, 非专家权重会被静默漏加载而不报错; 而新单测把 `_load_mxfp4_experts_weights` 整个替换成了记录器, 覆盖不到这类回归。
2. 加载期资源结构变化 (性能 / 内存, 低 - 中): 非专家权重现在会产生一次 D2H 拷贝并在 host 上多占数 GiB, 直到 checkpoint 读完; 显存峰值与修复前持平, 但 host 内存紧张或加载时间敏感的部署会感受到差异。作者未跑 speed benchmark, 判断「同样的字节、同样的顺序, 无速度影响」。
3. helper 重复实现 (可维护性, 低 - 中): `_own_if_runai_streamed` 是 RunAI ownership 逻辑在仓库中的又一份本地实现 (DeepSeek #32913、GLM5 #37370、NVFP4 #37371 各有一套), 若不收敛到 `runai_utils` 容易随版本漂移。
4. 验证层级不足 (测试, 低 - 中): 单测为纯 Python hermetic 测试 (`register_cpu_ci`), PR body 只提供了 gpt-oss-20b 单卡、`tp_size=1`、贪婪采样的离线对照, 没有 GSM8K 等精度评测, 也没有 120b 的端到端复现记录; 作者本人在 body 里也提示 "I have not run a GSM8K evaluation"。
5. CI 状态: PR Test (Base/Extra) 通过, AMD ROCm 10 的 run 显示失败, label 为 `run-ci/bypass-fastfail`; 考虑到改动只影响权重加载路径且不涉及 kernel, 需以最终 upstream 状态为准, 但仍是合入后需要留意的一条观察项。
 - 影响: 用户侧: 使用 `--load-format runai_streamer` 加载 gpt-oss-20b / 120b MXFP4 的部署, 从「静默权重损坏、所有请求返回非有限 logits」恢复为正常输出, 属于可用性级别的修复。问题在小规模下被掩盖——PR body 说明默认 buffer 下 12 GiB checkpoint 不会回绕, 因此真正受影响的是 gpt-oss-120b 以及把 `RUNAI_STREAMER_MEMORY_LIMIT` 压到最小 (如 1158266880) 的部署。

系统侧: 读取字节数与顺序不变, GPU 峰值显存与修复前持平, 加载期 host 内存增加数 GiB 并多一次 D2H 拷贝; 对 object-storage 分布式流式路径的影响面最大, 本地 safetensors 路径完全不受影响 (未标记的张量走原样返回分支)。

团队 / 生态侧: 为「RunAI 零拷贝视图 + 保留张量」这类反复出现的缺陷再一次提供了可复用范式——能即到即用的走生成器惰性消费, 必须保留的才复制, 且副本应落在 host 而不是 device 上; 同时为 `sglang.srt.model_loader.runai_utils` 的收敛增加了压力与一份参考实现。

- 风险标记：核心权重加载路径变更，非 mxfp4 路径未修复，生成器与正常加载的隐式耦合，host 内存峰值增加，helper 重复实现，缺少 GPU 端到端精度验证

关联脉络

- PR #32913 Fix RunAI streamer buffer race in DeepSeek weight loader: 同一类缺陷的首个修复 (PR body 明确引用)：RunAI 流式张量是复用 CPU buffer 的零拷贝视图，异步消费前推进迭代器会静默损坏权重；本 PR 的推理与它一致。
- PR #37370 Fix GLM5 RunAI streamer fused MLA weight ownership: 同类 ownership 修复：GLM5 缓存 q/kv-a 视图后拼接 fused MLA 权重导致损坏；该 PR 引入的 `sglang.srt.model_loader.runai_utils` 正是本 PR 建议后续收敛 helper 的目标模块。
- PR #37371 Fix NVFP4 RunAI streamer pending weight ownership: 尚未合并的同类修复 (对应 issue #37368)，本 PR body 引用它并约定：若它先合入，本地的 `_own_if_runai_streamed` 应迁移到共享 `runai_utils` 而不是成为第四份实现。
- PR #38988 Fix RunAI object-storage checkpoint index filtering: 同样位于 RunAI 对象存储权重加载链路，改动 `python/sglang/srt/model_loader/weight_utils.py` 等文件；本 PR 从该模块引入 `RUNAI_STREAMER_TENSOR_ATTR`，两条修复共同加固 RunAI 加载路径。