

PR #37413 完整报告

sgl-project/sglang

[AMD][DSV4] feat: enable fp8 two-pool unified_kv on gfx950

合并时间: 2026-09-14 17:49

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/37413>

执行摘要

- 一句话: DSV4 在 gfx950 启用 fp8 双池 KV, 容量提升 1.5x
- 推荐动作: 值得精读, 尤其是正在做 KV 量化、异构内存池或新硬件后端的人。建议按以下顺序看: 先读 `python/sglang/srt/mem_cache/deepseek_v4_memory_pool.py` 与 `unified_kv_kernels/layout.py` 建立行宽 / 两池的单一真相, 再读 `models/deepseek_v4.py` 的 `_forward_prepare` 看“用 caller buffer 决定布局分支”的设计, 最后看 `runtime.py` 与 `fused_qk_norm_rope_store.py` 理解写入 / 读取两侧的形态约束。

值得关注的设计决策有三个: (1) 把 E8M0 scale 在行内重复写两份、并用 512 B 2 的幂 stride 对齐 asm reader, 属于“为内核取数模式服务”的布局设计; (2) 所有未就绪组合一律显式拒绝并附原因与后续 PR 号, 而不是靠默认值静默降级; (3) 判据挂在 caller 传入的 buffer 上, 从结构上消除 prepare 与 backend 对布局的分歧。

如果要把它用于生产, 请先确认三件事: 目标场景的并发区间 (c=8 附近有 7.6% 解码损失)、是否需要 PD/HiCache/CP (当前均不可用)、以及是否有独立复测 (现有精度证据只有 GSM8K 的四组配对, 性能证据全部来自作者自测)。若只是评估设计而非上线, 本 PR 的价值在于它展示了一次“KV 行宽级重构”如何做到默认零影响、边界显式化。

功能与动机

PR body 明确指出了问题: unified_kv 下 DSv4 把整个 latent 都以 bf16 保存, `qk_nope_head_dim 448 + qk_rope_head_dim 64` 意味着“1024 B per token per layer”, 而“The nope half actually does not need that precision”。把 nope 按 1x64 tile 量化成 fp8 (带 E8M0 scale)、rope 保持 bf16, 行宽降到 640 B, “which measures out at 1.51x the KV capacity at identical memory on gfx950”。目标是在固定 HBM 下让更多 token 常驻, 支撑更长上下文与更高并发; 同时刻意把变更限定为默认关闭、gfx950 必需、其他架构 warn + 回退 bf16, 以保证既有路径行为不变 (“so nothing on the existing path changes”)。

实现拆解

本 PR 是“布局 + 写入 + 读取 + 接线 + 调优”一体化改造, 核心思路是把一个 bf16 latent 拆成行数一致、row index 语义共享的两个池, 让所有读写方按同一份行宽定义工作。

1. 布局与 sizing (`python/sglang/srt/mem_cache/deepseek_v4_memory_pool.py`、`python/sglang/kernels/ops/attention/dsv4/unified_kv_kernels/layout.py`): 新增 `DSV4_FP8_NOPE_ROW_BYTES = 512`、`DSV4_FP8_QUANT_TILE = 64` 与

check_two_pool_pair 守卫；DeepSeekV4UnifiedKVPool.__init__ 增加 fp8 形参，原 kv_buffer 改为 fp8 nope 行缓冲并新增 kv_buffer_rope (bf16 rope，行数与 nope 池严格相同)；dsv4_unified_row_bytes() 成为唯一行宽公式 (bf16 1024 B / fp8 640 B)，越界时抛 ValueError 而非 assert，理由写在代码注释里——“sizing runs under -O too, and a silently skipped check here overreports capacity”。pool_configurator.py 把行宽接入显存预算 (bytes_per_full_token 由 8918.00 降到 5945.00)，使容量报告与实际分配同源，这是 1.50x 数字可信的前提。

2. 写入侧三处 store：SWA scatter store_swa_into_unified 增加可选的 rope tensor / rope pool 参数，两个池按同一个 ring row index 落行，单字节行按 uint8 搬运以避免 E8M0 scale 的浮点转换；fused_qk_norm_rope_store.py 新增 _fp8_2buff_store，交给 aiter group-quant 融合内核，一次 launch 同时产出打包 fp8 nope 行与 bf16 rope 行，既有 packed 与 bf16 路径及其 module key 不变；compressor 侧新增 fp8_2buff 包装 (dsv4/compress.py、compressor_v2.py，仅对 512 宽 latent 构建)；JIT 模板 fused_norm_rope_v2.cuh 是 common code 的唯一改动点——新增一个 elif 分支，旧分支不动。
3. 读取侧两个内核：decode 走 aiter mla_a8w8 v4 nm asm (runtime.decode_fp8_2buff)，prefill 走 aiter opus (runtime.prefill_fp8_2buff，把 paged prefix 与本 chunk 的 flat extend 区用同一行布局寻址)；runtime.py 里补齐两池 / 行宽的启动前断言，并新增 decode_qo_indptr (qo_indptr 只依赖 token 数，改为每次 forward 建一次)。
4. 模型与后端接线 (python/sglang/srt/models/deepseek_v4.py、layers/attention/deepseek_v4_backend_hip_radix.py)：MQALayer._forward_prepare 新增 q_rope_out / k_nope_out / k_rope_out 三个 buffer 形参，并新增 fuse_prefill 与 fuse_verify_fp8 两个分支；所有判据都挂在 caller 传入的 buffer 上 (“Keyed off those buffers the same way fuse_prefill is, so the two arms cannot disagree about the layout”)，从结构上排除 prepare 与 backend 对布局判断不一致的可能。fp8 下 prefill/verify 的 kv 由 k_nope_out 顶替并保持 save_kv_cache，ring 写的时机由 backend 决定 (prefill 在 attention 之后、verify 在之前)。
5. 性能调优与测试配套：decode 路径删掉对空 segment 的 masked_fill_ (builders 产不出该形态)、跳过 group-quant 内核不需要的 kv.contiguous()、>40 token 时把 num_kv_splits 上限压到 4；prefill 分支同样去掉一次白拷贝。测试侧新增 8 个文件，覆盖行宽公式与两池分配 (test_dsv4_unified_fp8_pool.py)、SWA scatter、fused QK norm+RoPE、compressor store、decode/prefill 读数、backend prefill 接线与模型打包契约；其中两个纯 CPU 用例注册到 base-a-test-cpu，其余注册到 stage-b-test-1-gpu-small-amd-mi35x (非 gfx950 运行器会 skip)。

关键文件：

- python/sglang/srt/mem_cache/deepseek_v4_memory_pool.py (模块 KV 缓存池；类别 source；类型 data-contract；符号 dsv4_unified_row_bytes, DeepSeekV4UnifiedKVPool.get_unified_kv_rope, DeepSeekV4UnifiedKVPool.get_buf_infos, DeepSeekV4TokenToKVPool.get_unified_kv_rope)：双池布局与行宽公式的落点：新增 fp8 nope 池 + bf16 rope 池、dsv4_unified_row_bytes() 统一 sizing，并对单池假设的传输接口显式拒绝，是整条链路的地基。

- python/sglang/srt/models/deepseek_v4.py (模块 DSv4 模型; 类别 source; 类型 core-logic; 符号 MQALayer._forward_prepare, MQALayer.forward, fuse_verify_fp8, fuse_prefill) : 模型侧接线: _forward_prepare 新增三个 buffer 形参与 fuse_prefill / fuse_verify_fp8 分支, 决定打包对在 decode/prefill/verify 三种模式下的流向与 save_kv_cache 语义。
- python/sglang/kernels/ops/attention/dsv4/unified_kv_kernels/runtime.py (模块 内核运行时; 类别 infra; 类型 core-logic; 符号 decode_fp8_2buff, prefill_fp8_2buff, decode_qo_indptr, _scatter) : 读取侧入口: 新增 decode_fp8_2buff (aiter v4 nm asm)、prefill_fp8_2buff (aiter opus) 与 decode_qo_indptr, 并补齐两池 / 行宽的启动前守卫。
- python/sglang/srt/layers/attention/deepseek_v4_backend_hip_radix.py (模块 注意力后端; 类别 source; 类型 core-logic; 符号 DeepseekV4HipRadixBackend._forward_unified_kv) : _forward_unified_kv 在 fp8 下切换到 fp8 读内核, 并把同一对打包 buffer 既交给 attention 又用于 ring 写入——是“同一对数据两条消费路径”的枢纽。
- python/sglang/kernels/ops/attention/fused_qk_norm_rope_store.py (模块 融合归一化; 类别 infra; 类型 core-logic; 符号 _fp8_2buff_store, _token_identity_map, _HAS_GROUP_QUANT) : 写入侧核心封装: _fp8_2buff_store 一次 launch 产出打包 nope 行与 rope 行, 支持 strided kv 与 strided q_out, 是 Q 打包 (决定 decode reader 选择) 的来源。
- python/sglang/kernels/ops/attention/dsv4/unified_kv_kernels/layout.py (模块 行布局; 类别 infra; 类型 data-contract; 符号 DSV4_FP8_NOPE_ROW_BYTES, DSV4_FP8_QUANT_TILE, check_two_pool_pair) : 新增布局常量 (512 B 行、64 元素 tile) 与 check_two_pool_pair 守卫, 是分配侧与内核侧共享的 stride 单一真相。
- python/sglang/srt/model_executor/pool_configurator.py (模块 显存配置; 类别 source; 类型 data-contract) : 把双池行宽接入显存预算 (bytes_per_full_token / fixed SWA), 保证容量报告与实际分配同源, 是 1.50x 数字成立的前提。
- test/registered/unit/models/test_deepseek_v4_unified_fp8_q_pair.py (模块 打包契约; 类别 test; 类型 test-coverage; 符号 TestUnifiedFp8QPair, _run, _Harness, _Pool) : CPU 级契约测试: pin 住模型交给 backend 的“打包对”在 decode/prefill/verify/CP 各种模式下的形态, 以及未融合 store 与 CP 的拒绝语义。
- test/registered/e2e/dsv4/test_dsv4_unified_fp8_decode.py (模块 解码读取; 类别 test; 类型 test-coverage; 符号 TestUnifiedFp8Decode, _quantize_nope, _pack, _reference) : 用反量化参考 + 真实 index builder 输出 pin 住 decode 读数路径, 包括 cuda-graph pad 行落在保留 ring 槽、空 segment 返回 NaN、>40 token split 覆写等边角。
- test/registered/e2e/dsv4/test_dsv4_unified_fp8_backend_prefill.py (模块 预填写入; 类别 test; 类型 test-coverage; 符号 TestUnifiedFp8BackendPrefill, _Pool, _chunk, _run) : 补上此前没人跑过的“中间层”: _forward_unified_kv 如何在 fp8 下选分支并把打包对同时交给读内核与 ring 写, 专门防“rope 池留旧值”的静默回归。
- test/registered/e2e/dsv4/test_dsv4_unified_fp8_qk_norm_rope.py (模块 融合归一化; 类别 test; 类型 test-coverage; 符号 TestUnifiedFp8QkNormRope, _ref_norm_rope, _ref_tile_scales, _StoreCase) : pin 住 fused QK norm+RoPE store 的字节布局 (重复 E8M0 scale、strided kv、strided q_out) 以及 Q 在有无 rope buffer 时的两种返回形态。

- test/registered/unit/mem_cache/test_dsv4_unified_fp8_pool.py (模块 行宽计算; 类别 test; 类型 test-coverage; 符号 TestDSV4UnifiedRowBytes, TestDSV4UnifiedFp8PoolAllocation, dsv4_unified_row_bytes) : 把行宽公式钉死 (1024 $\rightarrow$ 640、0.625x、7 个 tile、50 B 填充) 并交叉校验分配字节数, 是容量收益正确性的直接守卫。
- python/sglang/kernels/jit/csrc/deepseek_v4/fused_norm_rope_v2.cuh (模块 JIT 内核; 类别 other; 类型 core-logic) : common code 的唯一改动点: JIT norm+rope 模板新增一个 fp8 双池 elif 分支, 其余分支保持不变。
- python/sglang/kernels/ops/attention/dsv4/unified_kv_kernels/env_gate.py (模块 特性开关; 类别 infra; 类型 configuration; 符号 is_unified_kv_fp8) : 新增 is_unified_kv_fp8() 门控, 把 SGLANG_DSV4_UNIFIED_KV_FP8、gfx95 与 is_unified_kv_triton() 三个条件收敛到一处, 是全部 fp8 分支的开关来源。
- python/sglang/srt/layers/attention/dsv4/compressor_v2.py (模块 压缩器; 类别 source; 类型 core-logic) : compressor 侧接线: c4/c128 压缩 latent 在 fp8 下改走双池写入, 是长上下文场景下压缩 KV 也享受容量收益的关键。
- python/sglang/srt/envron.py (模块 环境变量; 类别 source; 类型 configuration; 符号 SGLANG_DSV4_UNIFIED_KV_FP8) : 声明 SGLANG_DSV4_UNIFIED_KV_FP8 环境开关 (默认关闭), 决定特性对用户的可见入口。

关键符号: dsv4_unified_row_bytes, DeepSeekV4UnifiedKVPool.get_unified_kv_rope, DeepSeekV4TokenToKVPool.get_unified_kv_rope, MQALayer._forward_prepare, DeepseekV4HipRadixBackend._forward_unified_kv, decode_fp8_2buff, prefill_fp8_2buff, decode_qo_indptr, _fp8_2buff_store, fused_qk_norm_rope_swa_store, is_unified_kv_fp8, check_two_pool_pair, store_swa_into_unified

关键源码片段

python/sglang/srt/mem_cache/deepseek_v4_memory_pool.py

双池布局与行宽公式的落点: 新增 fp8 nope 池 + bf16 rope 池、dsv4_unified_row_bytes() 统一 sizing, 并对单池假设的传输接口显式拒绝, 是整条链路的地基。

```
# python/sglang/srt/mem_cache/deepseek_v4_memory_pool.py
# 行宽与 tile 尺寸由 kernels 侧的 layout 模块定义, 这里只做重导出:
# 分配行的池与消费行的内核必须共用同一份 stride 真相。
DSV4_FP8_NOPE_ROW_BYTES = layout.DSV4_FP8_NOPE_ROW_BYTES # 512 B
DSV4_FP8_QUANT_TILE = layout.DSV4_FP8_QUANT_TILE # 每 64 个元素一个 E8M0 scale
```

```
def dsv4_unified_row_bytes(
    qk_nope_head_dim: int, qk_rope_head_dim: int, fp8: bool
) -> int:
    """一个 token 在 unified_kv 中占用的字节数 (两个池相加) 。”
    if not fp8:
        # bf16 单池: 448 + 64 = 512 个 bf16 元素 = 1024 B / token / layer
        return (qk_nope_head_dim + qk_rope_head_dim) * 2
    # fp8 双池: nope 每元素 1 B, 外加每个 1x64 tile 一个 E8M0 scale;
```

```

# scale 在行内写两份，因为 asm reader 每个 tile 会读两次。
num_tiles = -(-qk_nope_head_dim // DSV4_FP8_QUANT_TILE) # 向上取整：448 -> 7
scale_bytes = 2 * num_tiles
# 这里必须是 raise 而不是 assert: sizing 在 python -O 下同样要生效，
# 静默跳过会把可用显存算多，进而按错误的行数去分配。
if qk_nope_head_dim + scale_bytes > DSV4_FP8_NOPE_ROW_BYTES:
    raise ValueError(
        f"fp8 nope row overflows: {qk_nope_head_dim} latent values at 1 B "
        f"{scale_bytes} B scales > {DSV4_FP8_NOPE_ROW_BYTES} B stride"
    )
# 512 B 定长 nope 行 + 128 B bf16 rope 行 = 640 B / token / layer (0.625x)
return DSV4_FP8_NOPE_ROW_BYTES + qk_rope_head_dim * 2

```

```

class DeepSeekV4UnifiedKVPool:

```

```

    """bf16 是单池 ``[rows, head_dim]``; fp8 是行数完全相同的两个池。

```

```

    row index 在两池中指向同一个 token (SWA ring 用
    ``req_pool_indices * swa_window + pos % swa_window``，压缩区用
    ``swa_pages + page_index``)，所以两边行数必须一一对应。
    """

```

```

def __init__(self, ..., swa_ring_size: int, fp8: bool = False): # 其余形参略
    self.fp8 = fp8
    self.rope_head_dim = qk_rope_head_dim
    self.head_dim = qk_nope_head_dim + qk_rope_head_dim
    bufs, rope_bufs = [], []
    # rows = swa_pages + padded_compress_rows, 两个池共用同一个 rows
    if self.fp8:
        bufs.append(
            torch.zeros(
                rows, DSV4_FP8_NOPE_ROW_BYTES,
                dtype=torch.float8_e4m3fn, device=device,
            )
        )
        rope_bufs.append(
            torch.zeros(
                rows, self.rope_head_dim,
                dtype=torch.bfloat16, device=device,
            )
        )
    else:
        bufs.append(
            torch.zeros(rows, self.head_dim,
                dtype=torch.bfloat16, device=device)
        )
        rope_bufs.append(None) # bf16 布局没有第二个池
    self.kv_buffer = bufs
    self.kv_buffer_rope = rope_bufs

```

```

def get_unified_kv_rope(self, local_layer_id: int) -> torch.Tensor:
    # rope 池只存在于 fp8 布局，调用方须先用 is_unified_kv_fp8() 分流
    assert self.fp8, "rope pool only exists under SGLANG_DSV4_UNIFIED_KV_FP8"
    return self.kv_buffer_rope[local_layer_id]

def get_buf_infos(self):
    if self.fp8:
        # 单池假设：一层一个指针 + 一个行宽只描述了 nope 池，
        # 消费者会搬走半行且毫无察觉，因此显式拒绝而不是给出错数据。
        # TODO(danli103): 等下游确实需要时再上报两个池。
        raise NotImplementedError(
            "get_buf_infos describes one pool per layer; the fp8 rope pool "
            "would be dropped (SGLANG_DSV4_UNIFIED_KV_FP8=1)."
        )
    ...

```

python/sglang/srt/models/deepseek_v4.py

模型侧接线：_forward_prepare 新增三个 buffer 形参与 fuse_prefill / fuse_verify_fp8 分支，决定打包对在 decode/prefill/verify 三种模式下的流向与 save_kv_cache 语义。

```

# python/sglang/srt/models/deepseek_v4.py —— MQALayer._forward_prepare 的 fp8 分流
unified = is_unified_kv_triton()
fp8_2buff = is_unified_kv_fp8()
is_decode = forward_batch.forward_mode.is_decode_or_idle()

```

```

fuse_verify = (
    envs.SGLANG_OPT_FUSED_QK_NORM_ROPE_VERIFY.get()
    and forward_batch.forward_mode.is_target_verify()
)
# verify 在 fp8 下像 prefill 一样打包 K，但保留 verify 自己的时序：
# ring 写在 attention 之前。判据全部挂在 caller 提供的 buffer 上，
# prepare 与 backend 因此不可能对布局产生分歧。
fuse_verify_fp8 = (
    fuse_verify and unified and fp8_2buff
    and k_nope_out is not None and k_rope_out is not None
)

```

```

# prefill 复用同一个 fused store：本 chunk 的 K 既是 attention 的 extend 区，
# 也是 attention 之后 ring 写入的源，所以只量化一次、两边共用这对 buffer。

```

```

fuse_prefill = (
    unified and fp8_2buff
    and k_nope_out is not None and k_rope_out is not None
    and not is_decode
    and not forward_batch.forward_mode.is_target_verify()
)
do_fused_qk_norm_rope = (
    unified and (is_decode or fuse_verify or fuse_prefill)
) or (not unified and self.use_fused_qk_norm_rope)

```

```

# …… (中间省略 call kernel 的部分，调用处会带上 fp8_2buff / swa_rope_cache /

```

```

# k_nope_out / k_rope_out / q_rope_out)

swa_rope_cache = None
if unified and fuse_verify:
    # verify 读 ring 的姿势与 decode 相同，但自带 per-token slot map;
    # kv 是 qkv_a 的 strided slice，未融合路径要在这里显式 contiguous() 出来。
    kv = kv.contiguous()
    swa_cache, swa_loc = None, None
    swa_page_size, bf16_store = 1, not fuse_verify_fp8
elif unified and fuse_prefill:
    # 这次不写 ring：这些行是本 fwd 的 extend 区，prefix 池必须保持
    # attention 期望的形态，ring 由 backend 在 attention 之后写入。
    # kv 继续保持 qkv_a 的 stride —— fp8 下内核只读它，并且按 stride
    # 参数取行，所以原来的 contiguous() 是纯粹的白拷贝（本 PR 把它删掉）。
    swa_cache, swa_loc = None, None
    swa_page_size, bf16_store = 1, False
elif unified:
    swa_cache = token_to_kv_pool.get_unified_kv(self.layer_id)
    # swa_loc 与 layer 无关：backend 每次 forward 计算一次并缓存在 metadata
    swa_loc = attn_backend.get_unified_swa_loc(forward_batch)
    swa_page_size, bf16_store = 1, not fp8_2buff
    if fp8_2buff:
        # decode 的第二个写入目标：bf16 rope 池
        swa_rope_cache = token_to_kv_pool.get_unified_kv_rope(self.layer_id)

if unified and (fuse_prefill or fuse_verify_fp8):
    # 打包后的 nope 半行沿 kv 槽返回，attention 把它当 attn_k,
    # save_kv_cache 保持打开让 backend 完成 ring 写；rope 半行去了
    # caller 的 buffer，这里没有第二个返回槽。prefill 在 attention 之后写、
    # verify 在之前写，但两者读的是同一对 buffer。
    kv = k_nope_out
elif not (unified and fuse_verify):
    kv = None

```

test/registered/unit/models/test_deepseek_v4_unified_fp8_q_pair.py

CPU 级契约测试：pin 住模型交给 backend 的“打包对”在 decode/prefill/verify/CP 各种模式下的形态，以及未融合 store 与 CP 的拒绝语义。

```

# test/registered/unit/models/test_deepseek_v4_unified_fp8_q_pair.py
# 这些用例 pin 的是“交给 backend 的是不是一个打包对”，而不是内核算术：
# decode 只需要打包的 Q（K 已经在 ring 里）；prefill 自己就是 KV 源，
# 因此还要一对打包的 K；verify 两者都要——像 decode 一样读、像 prefill 一样写。

```

```

class _Harness(deepseek_v4.MQALayer):
    def _forward_prepare(
        self, x, positions, forward_batch, attn_backend,
        q_out=None, x_quant=None, q_rope_out=None,
        k_nope_out=None, k_rope_out=None,
    )

```

```

):
# 记下 caller 传入的 buffer, 用于断言“store 填的”就是“attention 读的”
self.prepare_kwargs = dict(
    q_out=q_out, q_rope_out=q_rope_out,
    k_nope_out=k_nope_out, k_rope_out=k_rope_out,
)
q_out.zero_()
# 与 prefill 分支一致: 打包后的 nope 半行沿 kv 槽返回,
# 这正是调用方把 save_kv_cache 打开的原因
return q_out, k_nope_out

```

```

class TestUnifiedFp8QPair(unittest.TestCase):
    def test_fp8_decode_hands_the_backend_a_packed_pair(self):
        layer, call = _run(fp8=True)

        q, q_rope = call["q"], call["q_rope"]
        self.assertEqual(q.dtype, torch.float8_e4m3fn)
        # 行宽取自 pool 而不是 head_dim: fixture 里两者被刻意设成不同值,
        # 否则“误读 self.head_dim”的回归会照样通过
        self.assertEqual(tuple(q.shape), (TOKENS, N_LOCAL_HEADS, NOPE_ROW_BYTES))
        self.assertEqual(q_rope.dtype, torch.bfloat16)
        # asm reader 把两者当扁平 buffer 走, 没有任何 stride 参数
        self.assertTrue(q.is_contiguous())
        self.assertTrue(q_rope.is_contiguous())
        # 同一对 buffer 也送进了 store, 否则根本没有东西把它们写出来
        self.assertIs(layer.prepare_kwargs["q_out"], q)
        self.assertIs(layer.prepare_kwargs["q_rope_out"], q_rope)

    def test_fp8_target_verify_gets_the_packed_pair(self):
        """verify 像 decode 一样读 ring, 又像 prefill 一样喂 ring"""
        layer, call = _run(fp8=True, mode=ForwardMode.TARGET_VERIFY)

        # 打包 Q 是选中 decode reader 而不是 Triton reader 的开关
        self.assertEqual(call["q"].dtype, torch.float8_e4m3fn)
        k, k_rope = call["k"], call["k_rope"]
        self.assertEqual(k.dtype, torch.float8_e4m3fn)
        self.assertEqual(tuple(k.shape), (TOKENS, NOPE_ROW_BYTES))
        self.assertIs(layer.prepare_kwargs["k_nope_out"], k)
        self.assertIs(layer.prepare_kwargs["k_rope_out"], k_rope)
        # 与 prefill 不同, ring 写在 attention 之前, 但标志位与 buffer 是同一对
        self.assertTrue(call["save_kv_cache"])

```

评论区精华

本 PR 没有行内 review 评论, 评审信号只有一条 **APPROVED: LGTM** (HaiShaw) 与 4 条 issue 级评论; 真正有交锋价值的内容几乎全部由作者写在 PR body 里, 属于“作者自我 review + 一条 LGTM 放行”的评审形态。

值得记录的讨论要点（引自 PR body / 评论）：

HaiShaw: /tag-and-rerun-ci; 随后 @ispobock @Fridge003 @hnyls2002 @xiezhq-hermann @hzh0425 @hebiao064 @BBuf @ShangmingCai Please have a review.

xiezhq-hermann: @DarkSharpness can you help take a look?

HaiShaw: PR Test Base & PR Test Extra all pass. (review 结论: LGTM)

作者主动标注最差格子: “The cell to flag is 8192/1024 at c=8 under TP=8: 7.6% less throughput, ITL +1.18 ms. That is the worst of the 14 TP=8 cells ... The leftover gap is the decode reader, not anything this PR still owns.” 结论是 aiter 的 mla_a8w8 asm 内核按 64 q head 调优, TP=8 只剩 16 个本地 head 时受占用率限制, “Closing the TP=8 gap needs an mla_a8w8 variant for 16 q heads on the aiter side; that is a separate PR.”

能力边界的处理方式: “PD disaggregation and HiCache offload are refused for now because both assume one pool per layer.” 代码里落实为 get_buf_infos / get_contiguous_buf_infos / get_unified_swa_ring_buf_infos 抛 NotImplementedError, 并带 TODO(danli103) 与后续 PR 号。

被放弃的方案: “A Triton rewrite was measured and was slower at every shape, so it was dropped.”

测试落位也经过一轮修正 (commit f28857c): “#37436 rejects new files under test/registered/amd/. These stay on the mi35x stage-b suite, so e2e/dsv4 is the kind that keeps that registration.”

未在讨论中解决、只能靠后续 PR 收口的问题: TP8 低并发解码回退、PD (#39166)、HiCache (#37778)、DSpark MTP (#38901)、HiSparse 与 PCP。

- 1.50x 容量收益与 GSM8K 无回归的验证口径 (design): 评审以一条 LGTM 接受作者自证的数据, 没有对测量方法提出独立质询。
- TP8 低并发解码回退的归因与归属 (performance): 不在本 PR 内修复, 结论是“Closing the TP=8 gap needs an mla_a8w8 variant for 16 q heads on the aiter side; that is a separate PR”, 风险被显式转嫁给外部依赖。
- PD/HiCache 单池假设的显式拒绝 (correctness): 以“宁可报错不可静默丢 rope”为原则冻结能力, PD 与 HiCache 分别由 #39166、#37778 补齐。
- 未支持组合用运行时错误而非静默回退 (testing): 边界以可测试的错误语义固化, 避免组合爆炸变成静默错算。
- 测试目录规约与 CI 桩对象的兼容性修补 (testing): 通过类级默认 _unified_kv_fp8 = False 与新目录规约对齐, 代价是部分测试在非 gfx950 运行器上整体 skip。

风险与影响

- 风险: 兼容性与回归

- 默认关闭 + gfx95 门控 (env_gate.is_unified_kv_fp8() 要求 SGLANG_DSV4_UNIFIED_KV_FP8 且 gfx95)，bf16 路径在所有写入 / 读取点都保留了独立分支，测试也逐条 pin 了“bf16 不变”断言（如 test_bf16_decode_still_gets_one_plain_tensor、test_the_bf16_arm_never_touches_the_rope_pool、test_bf16_single_pool_unchanged），因此存量用户风险低。风险集中在开启开关之后的组合空间。
- 未支持组合 (PD、HiCache offload、DSA prefill CP、HiSparse、PCP) 靠 NotImplementedError / AssertionError 显式拒绝，不会静默退化；但这依赖调用方先经 is_unified_kv_fp8() 分流——get_unified_kv_rope() 内部是 assert self.fp8，门控不一致时会以断言形式炸在运行时，而非给出明确语义错误。

性能

- 作者自测最差格子为 TP8、ISL/OSL 8k/1k、concurrency=8: ttt 比值 0.924 (bf16 4837 → fp8 4470)，ITL +1.18 ms；14 个 TP8 格子中其余都在 4.3% 以内，concurrency ≥ 16 全部在 3.2% 以内，DP8 表基本落进噪声。根因是 aiter mla_a8w8 asm 内核按 64 q head 调优、TP=8 下只剩 16 个本地 head，属于外部依赖问题，本 PR 无法消除。
- 本 PR 已顺手做了若干调优（删 masked_fill_、decode_qo_indptr 复用、去掉两处 kv.contiguous()、num_kv_splits 上限 4），但这些都是作者自述，未见独立 benchmark 复核。

显存与数据正确性

- 容量收益完全建立在 dsv4_unified_row_bytes() 与两池实际分配一致之上；test_fp8_pool_bytes_match_the_sizing_row_width 做了交叉校验，test_fp8_saves_exactly_three_eighths 钉住 $640/1024 = 0.625$ 。若未来 head_dim 或 tile 尺寸变化，ValueError（而非 assert）保证 -O 下也能拦住。
- 两池行数必须相等，row index 才在两池中语义一致；check_two_pool_pair 与测试里的“pool rows differ”断言提供了启动前保护。
- get_buf_infos 在 fp8 下直接抛错，说明有人用时会立刻失败而不是搬半行——这是好的防御，但也意味着 PD 场景在 #39166 落地前完全不可用。

精度

- nope 走 1x64 tile 的 E8M0 scale，误差上限约为该 tile absmax 的 2^{-4} （测试用 $0.07 * \text{tile_absmax}$ 作为 dequant 判据）；PR 只给出 GSM8K 1319 题的四组配对结果（0.940/0.945/0.952/0.947 对 0.949/0.948/0.947/0.946），长上下文端到端精度未做量化评测。

测试与 CI 覆盖

- 所有双池硬件用例都注册在 stage-b-test-1-gpu-small-amd-mi35x，在 mi300 等非 gfx950 运行器上会被整体 skip，等于覆盖强绑特定硬件池。
- PR body 的 CI 状态里“Latest PR Test (AMD ROCm 10)”标记为失败 (:x:)，而 issue 评论只声明 PR Test Base & PR Test Extra 通过；合并依据没有覆盖 AMD ROCm 10 这条线。

- 3594 行新增、21 个文件、13 个 commit，但只有一条 LGTM 评审、零行内评论，评审深度实质等同于作者自证 + CI 绿灯。

过程风险

- PR 自带 checklist 全部未勾选（格式化、单测、文档、benchmark），但 body 里实际附了完整可复现实验步骤与基准数据，说明流程痕迹与实际证据不一致。
- 代码里留有两处 TODO(danli103) 与显式拒绝路径，属于“已知未完成”，若后续 PR 未落地会成为长期分支。
- 影响：对用户
- 在 MI355X / gfx950 上跑 DSV4 的用户，打开 SGLANG_DSV4_UNIFIED_KV_FP8 即可在字节级相同的 CLI 下获得约 1.50x 的 KV 容量（TP8: 16,126,208 → 24,223,744 full_token; TP8+DP8+MTP 为 13,612,288 → 20,424,448），代价是低并发解码 1%~7.6% 吞吐；默认关闭，存量部署零影响。
- 需要长上下文 / 高并发但可以接受轻微解码损失的线上场景（例如大并发 68k 输入）收益明显：作者测的 68k/200 各并发点 ttt 比值 0.980~1.002，基本无损。

对系统

- KV 行宽从 1024 B 变为 640 B，改变了显存预算公式（bytes_per_full_token 8918.00 → 5945.00，实际比值 0.6666 而非行宽的 0.625，因为 indexer 项不随行宽缩放），直接影响 pool sizing、full_token 报告与容量规划；容量计算与分配同源是这套收益成立的前提。
- row index 语义从“一个池的行”变为“两个池共享的行”，所有池消费者（PD 传输、HiCache offload、region buffer）都必须重新审视单池假设；本 PR 以显式拒绝换来安全性，把工作留给 #39166 / #37778。
- 引入了对 aiter（v4 nm asm decode、opus prefill、group-quant 融合 store）与 gfx950 的硬依赖，非 gfx950 平台只有警告回退路径。

对团队

- 功能线被拆成清晰的后续清单（DSpark #38901、HiCache #37778、PD #39166、HiSparse 与 PCP 待定），短期对外能力是“仅在 gfx950 且关闭 PD/HiCache/CP 的前提下可用”。
- 测试资产（8 个新文件，覆盖写入 / 读取 / 接线 / 拒绝路径）为后续在每个新能力上扩展提供了模版，尤其是用 sentinel 值区分“没写”和“写了 0”、用反量化参考隔离“接线错误 vs fp8 舍入”的做法。
- 风险标记：核心 KV 布局变更，默认关闭的硬件专属特性，TP8 低并发吞吐回退，PD/HiCache 仅拒绝未支持，测试强绑 mi35x（非 gfx950 全 skip），评审覆盖薄（单条 LGTM）

关联脉络

- PR #37506 [unified-memory] PD disaggregation for every unified pool shape: 同一条 unified_kv 内存池线的上一环：它让各种 unified pool 形态都能做 PD 传输，本 PR 正是因为 fp8 双池下打破了“一层一池”假设，才必须在 get_contiguous_buf_infos/get_unified_swa_ring_buf_infos 上显式拒绝 PD。

- PR #37418 [unified-memory] Enable prefill cuda-graph capture: 同一 unified 内存池功能线, 涉及 pool allocator、attention backend 与 prefill 图捕获的协同; 本 PR 的 decode 测试也专门覆盖了 cuda-graph pad 行落在保留 ring 槽的行为。
- PR #37382 [NPU] Support DSV4 host memory cache management: 同为 DSV4 内存池扩展 (memory_pool_host / schedule_batch / unified_radix_cache), 与本 PR 共享“池形态变化会波及 HiCache 卸载与调度”的问题域; 本 PR 的 HiCache 只能拒绝, 由 #37778 继续。
- PR #37436 测试目录规约调整 (commit f28857c 引用): 该 PR 拒绝在 test/registered/amd/ 下新增文件, 直接导致本 PR 的双池 gfx950 用例改到 test/registered/e2e/dsv4/ 并注册 mi35x stage-b 套件。
- PR #39166 PD 双池支持 (PR 正文 Follow-up 引用的后续 PR): 本 PR 在 mem_cache 中显式拒绝 PD 传输并留下 TODO(danli103), 正文列出“PD is now supported in #39166”, 是本特性解除限制的直接后续。
- PR #37778 HiCache 双池支持 (PR 正文 Follow-up 引用的后续 PR): 同理, HiCache offload 因单池假设被拒绝, 正文写明“HiCache is further supported in #37778”, 两者共同决定本 PR 的能力边界。