PR #30531 完整报告

sgl-project/sglang

[DSA] Skip indexer KV cache for skip-topk layers

合并时间：2026-08-17 17:02

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30531

执行摘要

- 一句话：skip-topk 层免建 Indexer 并省 index-K 缓存，KV 容量提升约 16%
- 推荐动作：值得精读。核心学习点是“elision 门控 + 预算联动 + 读写跳过”三段式设计：先由 _should_elide_dsa_index_k 统一判定哪些场景可以省内存，再让池尺寸计算（_compute_dsa_indexer_cell_size）、池构建（skip_topk_layers 掩码）和缓存读写（0 行占位 buffer）三处共享同一判定，避免口径漂移。另一个值得关注的点是 CP 分片下用 max_owned + 1 的保守上界保证各 rank 容量一致，以及 EAGLE draft 必须 allocate_all_layers=True 全额计费。

功能与动机

DSA（DeepSeek Sparse Attention）模型中大量 skip-topk 层复用上一层 topk 结果，既不写自己的 index-K，也无需独立的 Indexer 模块。mmangkad 的 A/B 数据说明：GLM-5.2 的 index_topk_freq=4 / index_skip_topk_offset=3 使得 78 层中只有 21 层是 active indexer 层，57 层是 skip-topk。原 #30310 已通过 indexShare 逻辑把 KV 池扩大 15%，本 PR 更进一步把这些层未使用的 indexer 权重（约 0.98 GB/rank）和 index-K 缓存全部省下来，转给 KV 池使用。同时 #30310 曾因与 HiCache 不兼容触发 IMA 崩溃（dongyibo 报障），本轮用显式门控解决。

实现拆解

实现分五步展开，核心是“elision 门控 + 预算联动 + 读写跳过”三处联动：

1. 模型构建侧跳过 Indexer 构造：python/sglang/srt/models/deepseek_v2.py 在 DeepseekV2Attention.__init__ 中先计算 skip_topk / next_skip_topk，仅当 not self.skip_topk or is_nextn 时才实例化 Indexer，其余层 self.indexer = None。skip-topk 语义收敛到 model_config.dsa_layer_skips_topk()，cli_factor（LongCat 模式）分支从模型文件下沉到该配置函数，避免两处逻辑漂移。
2. 配置辅助函数统一判定：python/sglang/srt/configs/model_config.py 的 dsa_layer_skips_topk 新增 cli_factor > 1 分支（按 layer_id % cli_factor != 0 判定），并同步更新 get_num_indexer_layers 文档，明确 capturer 槽位与 Indexer 模块解耦。
3. 缓存池构建与尺寸计算联动：kv_cache_configurator.py 新增 _should_elide_dsa_index_k() 门控（hisparse、draft worker、HiCache、PD disagg 均禁用 elision），_build_dsa_kv_pool 在允许时向 DSATokenToKVPool 传入

skip_topk_layers 掩码；memory_pool.py 的 DSATokenToKVPool.__init__ 新增该参数并默认全 False。

4. 池尺寸计算只对 active 层计费：pool_configurator.py 把原内联的 indexer 开销计算抽成 _compute_dsa_indexer_cell_size()，正常路径仅统计非 skip-topk 层；CP 层分片下取所有分片 max_owned + 1 的保守上界保证各 rank 容量一致；EAGLE draft 用 allocate_all_layers=True 全额计费，避免 target 折扣后 draft 预算不足。
5. index-K 缓存读写跳过 0 行层：index_key_cache.py 的 _layer_num_pages 对 skip-topk 层返回 0 行占位 buffer；move / cpu_copy / load_cpu_copy 遇到 0 行跳过；_item_len 对 0 行返回 0 并接入 state_buf_infos。dsa_cache_layer_split.py 的 LayerSplitIndexKeyCache 改为先判 _is_layer_owned 再调用 super()._layer_num_pages，使 CP 分片与 skip-topk 两个条件叠加生效。
6. 权重加载跳过未构造模块：deepseek_weight_loader.py 先收集含 .indexer. 的参数前缀集合，加载时若某个 .indexer. 权重所属前缀不在集合中则跳过，避免因模块缺失导致 key 不匹配报错。
7. 测试配套：test_pool_configurator.py 新增 _configure_dsa_model 真实 DSA 配置（GlmMoeDsaForCausalLM、index_topk=2048、index_head_dim=128），覆盖 EAGLE draft 全额 indexer 计费、HiCache override 后全量计费、PD 模式下全量计费等策略；test_hisparse_pool_configurator.py 改为通过 runtime_context 发布 server args override。曾新增长上下文 bench 测试文件，最后提交中移除。

关键文件：

- python/sglang/srt/model_executor/pool_configurator.py（模块 池配置；类别 source；类型 data-contract；符号 _compute_dsa_indexer_cell_size, _compute_cell_size）：内存池尺寸计算核心文件。抽取 _compute_dsa_indexer_cell_size，让 indexer 开销只按 active 层计费，并处理 CP 分片与 EAGLE draft 两种特殊预算；EAGLE 分支把 target 的 KV 与 indexer 分开缩放，避免 draft 全额 index-K 被 target 的 elision 折扣掉。
- test/registered/unit/model_executor/test_pool_configurator.py（模块 单元测试；类别 test；类型 test-coverage；符号 _configure_dsa_model, TestDSAIndexerAllocationPolicy, test_dsa_draft_full_indexer_cost_does_not_exceed_budget, test_resolved_hicache_override_prices_every_indexer_layer）：测试主配套，新增真实 DSA 配置 helper 与 5 个策略用例，把 HiCache override、PD 模式、EAGLE draft 全额计费这三个边界条件用单元测试钉死，防止门控回归。
- python/sglang/srt/models/deepseek_v2.py（模块 模型构建；类别 source；类型 core-logic；符号 Indexer, skip_topk, next_skip_topk）：模型构建核心改动：self.indexer = None 兜底，仅对非 skip-topk 层或 nextn 层构造 Indexer，从源头省掉 57/78 层的权重与显存；cli_factor 判定逻辑移除后统一走 dsa_layer_skips_topk。
- python/sglang/srt/mem_cache/index_key_cache.py（模块 索引缓存；类别 source；类型 core-logic；符号 _layer_num_pages, _item_len, move, cpu_copy）：index-K 缓存读写核心：skip-topk 层分配 0 行占位 buffer，move/cpu_copy/load_cpu_copy 跳过 0 行层，state_buf_infos 通过 _item_len 对 0 行返回 0；同时修复 load_cpu_copy 误用 buffer[0].device 的隐患。

- python/sglang/srt/mem_cache/kv_cache_configurator.py（模块 缓存配置；类别 source；类型 core-logic；符号 _should_elide_dsa_index_k, _build_dsa_kv_pool）：elision 门控所在地：_should_elide_dsa_index_k 统一判定 hisparse、draft worker、HiCache、PD disagg 四类场景，_build_dsa_kv_pool 据此向池传入 skip_topk_layers 掩码，是整套优化的安全边界。
- python/sglang/srt/configs/model_config.py（模块 模型配置；类别 source；类型 configuration；符号 dsa_layer_skips_topk, get_num_indexer_layers）：skip-topk 判定被收敛为唯一入口：dsa_layer_skips_topk 新增 LongCat 的 cli_factor 分支，模型层与池构建都调用它，避免判定口径在多处漂移。
- python/sglang/srt/models/deepseek_common/deepseek_weight_loader.py（模块 权重加载；类别 source；类型 core-logic；符号 do_load_weights）：skip-topk 层不再有 Indexer 模块后，加载 checkpoint 时需跳过对应 indexer 权重，否则 key 不匹配报错；通过收集 indexer_present_prefixes 决定是否跳过。
- python/sglang/srt/mem_cache/memory_pool.py（模块 内存池；类别 source；类型 data-contract；符号 DSATokenToKVPool）：DSATokenToKVPool 新增 skip_topk_layers 构造参数（默认全 False），作为 index-K 0 行分配与模型层掩码之间的数据契约。
- python/sglang/srt/mem_cache/dsa_cache_layer_split.py（模块 分层缓存；类别 source；类型 core-logic；符号 LayerSplitIndexKeyCache）：CP 层分片与 skip-topk 两个条件需叠加生效：LayerSplitIndexKeyCache._layer_num_pages 改为先判 owned 再调 super，state_buf_infos 改用 _item_len 处理 0 行层。
- test/registered/unit/model_executor/test_hisparse_pool_configurator.py（模块 单元测试；类别 test；类型 test-coverage；符号 _compute_cell_size）：HiSparse 场景的配套测试：把 server args override 改为通过 runtime_context 发布，覆盖 enable_hisparse 与 host_to_device_ratio 对 indexer 计费的影响。

关键符号：_compute_dsa_indexer_cell_size, _should_elide_dsa_index_k, dsa_layer_skips_topk, _layer_num_pages, _item_len, _build_dsa_kv_pool

关键源码片段

python/sglang/srt/model_executor/pool_configurator.py

内存池尺寸计算核心文件。抽取 _compute_dsa_indexer_cell_size，让 indexer 开销只按 active 层计费，并处理 CP 分片与 EAGLE draft 两种特殊预算；EAGLE 分支把 target 的 KV 与 indexer 分开缩放，避免 draft 全额 index-K 被 target 的 elision 折扣掉。

```
def _compute_dsa_indexer_cell_size(
    self,
    *,
    kvc: KVCacheConfigurator,
    num_layers: int,
    allocate_all_layers: bool = False,
) -> int:
    """计算单 token 的 DSA indexer KV 缓存字节开销。
```

```
allocate_all_layers=True 时（如 EAGLE draft 模型）按全部层计费，
因为 draft 不会复用 skip-topk 层的 index，必须持有完整 index-K。
"""
index_head_dim = get_dsa_index_head_dim(kvc.model_config.hf_config)
# 每 token 存储：index_head_dim 个 FP8 值 + 每 quant_block_size 一个 FP32 scale
indexer_size_per_token = (
    index_head_dim + index_head_dim // DSATokenToKVPool.quant_block_size * 4
)
element_size = torch._utils._element_size(
    DSATokenToKVPool.index_k_with_scale_buffer_dtype
)
memory_config = get_memory()
indexer_ratio = 1
if memory_config.enable_hisparse:
    # HiSparse 把部分 index-K 放 host 侧，device 预算按 host_to_device_ratio 折算
    from sglang.srt.mem_cache.sparsity import parse_hisparse_config

    indexer_ratio = parse_hisparse_config(kvc.server_args).host_to_device_ratio

from sglang.srt.mem_cache.kv_cache_configurator import _should_elide_dsa_index_k

if allocate_all_layers or not _should_elide_dsa_index_k(
    is_draft_worker=kvc.is_draft_worker
):
    num_indexer_layers = num_layers
else:
    # 正常推理路径：skip-topk 层复用上一层 topk，无需自己的 index-K 槽位
    active_indexer_layers = [
        layer_id
        for layer_id in range(kvc.layer_info.start_layer, kvc.layer_info.end_layer)
        if not dsa_layer_skips_topk(kvc.model_config.hf_config, layer_id)
    ]
    from sglang.srt.layers.cp.utils import (
        get_glm_dsa_cp_layer_shard_info,
        get_layer_shard_range,
    )

    _, shard_size = get_glm_dsa_cp_layer_shard_info(kvc)
    if shard_size > 1:
        # CP 层分片下每张卡只持有部分层；取所有分片 active 层数最大值 + 1
        # 作为统一预算，保证任意 rank 的 token 容量一致
        active_set = set(active_indexer_layers)
        max_owned = 0
        for rank in range(shard_size):
            start, end = get_layer_shard_range(rank, shard_size, num_layers)
            max_owned = max(
                max_owned,
                sum(
                    kvc.layer_info.start_layer + i in active_set
```

```
                    for i in range(start, end)
                ),
            )
            num_indexer_layers = max_owned + 1
        else:
            num_indexer_layers = len(active_indexer_layers)

    return int(
        indexer_size_per_token * num_indexer_layers * element_size * indexer_ratio
    )
```

python/sglang/srt/mem_cache/index_key_cache.py

index-K 缓存读写核心：skip-topk 层分配 0 行占位 buffer，move/cpu_copy/load_cpu_copy 跳过 0 行层，state_buf_infos 通过 _item_len 对 0 行返回 0；同时修复 load_cpu_copy 误用 buffer[0].device 的隐患。

```
def _layer_num_pages(self, layer_idx: int, num_pages: int) -> int:
    # skip-topk 层永远不会向自己的 index-K 槽位写入（复用上一层 topk），
    # 因此分配 0 行占位 buffer；buffer 列表仍按层对齐，索引逻辑不变。
    return 0 if self.pool.skip_topk_layers[layer_idx] else num_pages


def move(self, tgt_loc: torch.Tensor, src_loc: torch.Tensor) -> None:
    if tgt_loc.numel() == 0:
        return
    tgt_loc_flat = tgt_loc.view(-1).long()
    src_loc_flat = src_loc.view(-1).long()
    for index_k in self.buffer:
        if index_k.shape[0] == 0:
            # 0 行占位层没有页可搬移，直接跳过，避免空张量索引越界
            continue
        index_k[tgt_loc_flat] = index_k[src_loc_flat]


def _item_len(self, layer_idx: int) -> int:
    # 0 行层（skip-topk，或 CP 分片下非本卡持有层）没有 item，返回 0
    buf = self.buffer[layer_idx]
    return 0 if buf.shape[0] == 0 else buf[0].nbytes


def state_buf_infos(self):
    layer_num = self.pool.layer_num
    data_ptrs = [self.buffer[i].data_ptr() for i in range(layer_num)]
    data_lens = [self.buffer[i].nbytes for i in range(layer_num)]
    # 旧实现直接取 buffer[i][0].nbytes，对 0 行层会越界；改走 _item_len
    item_lens = [self._item_len(i) for i in range(layer_num)]
    return data_ptrs, data_lens, item_lens
```

评论区精华

Review 中最有价值的交锋集中在兼容性与负载器变更：

- dongyibo 报障：--enable-hierarchical-cache 下触发 scheduler 异常（IMA 问题），这是 #30310 曾失败的直接原因。结论是本 PR 的 _should_elide_dsa_index_k 把 enable_hierarchical_cache、enable_hisparse、PD disagg 和 draft worker 全部列为禁止 elision 的条件，从根上规避。
- b8zhong 询问"rebase #30310 是否兼容 HiCache"，dongyibo 表示当时启动即触发 IMA 未再跟进；本 PR 用门控显式声明了兼容边界。
- Fridge003 追问 deepseek_weight_loader.py 新增逻辑原因，mmangkad 答复：skip-topk 层不再创建 Indexer 模块，加载时需跳过这些层的 indexer 权重。
- Fridge003 建议把长上下文 bench 抽成可复用抽象，mmangkad 先拆到 python/sglang/test/kits/long_context_bench_kit.py，最终提交又移除了该文件。
- HiCache 不兼容与 IMA 崩溃 (correctness): 本 PR 将 enable_hierarchical_cache、enable_hisparse 纳入 _should_elide_dsa_index_k 禁止条件，HiCache 场景回退全量 index-K 分配。
- 30310 rebase 是否兼容 HiCache (question): 30531 用显式门控声明不兼容边界，回退逻辑覆盖 HiCache，比简单 rebase 更可靠。
- weight loader 新增跳权逻辑的原因 (question): 接受该设计，跳权基于 indexer_present_prefixes 集合判定。
- 长上下文 bench 的可复用抽象 (design): 已按建议抽象，但后续提交删除 bench 测试文件，保留 nightly 长上下文用例由其它 PR 覆盖。
- pool_configurator 冗余注释清理 (style): 注释已清理。

风险与影响

- 风险：风险集中在门控覆盖与 0 行 buffer 的读写边界：

1. 门控遗漏导致内存超支：_should_elide_dsa_index_k 目前覆盖 hisparse、draft worker、HiCache、PD disagg 四类场景，但依赖 get_memory() / get_disagg() 运行时上下文；若未来新增模式（如 disagg + HiCache 组合的某种传输优化）未同步更新门控，会按缩减预算分配却实际全量写入 index-K，造成 OOM 或越界。
2. 0 行 buffer 的潜在越界：IndexKeyCache 的 store_quantized / get_k_* 系列没有显式防御 0 行 buffer，语义上 skip-topk 层不会读写自己的 index-K，但若层索引错位或 CP 分片边界覆盖不全，空张量会触发索引报错。load_cpu_copy 顺带修正了 buffer[0].device → buffer[layer_id].device，说明该处边界此前就有隐患。
3. draft 计费过度保守：EAGLE draft 用 allocate_all_layers=True 全额计费，比实际可能略保守，但保证了 draft 侧 index-K 不欠账，属于可接受的取舍。
4. CP 分片 max_owned + 1：+1 是保守上界，会浪费少量显存，但换来各 rank token 容量一致，符合现有 token 池统一预算模型。
5. 权重加载依赖 .indexer. 子串匹配：若未来 indexer 内层命名变化（如嵌套 prefix）可能漏跳或误跳，建议补充断言。 - 影响：影响范围集中在 DSA 模型（GLM-5.2 等）在

Blackwell/GB300 上的长上下文推理：

- 用户侧：4×GB300 实测 max_total_num_tokens 从 2,574,848 提升到 3,002,752（+16.62%），bytes/token 从 55,225 降到 47,702（-13.62%），每卡权重占用 -0.98 GB；128K 长输入场景直接受益，gsm8k 等质量指标由 sgl-eval 验证保持（pass@1 90.83%）。
- 系统侧：HiCache、hisparse、PD 分离部署用户自动回退全量分配，行为与 #30310 之前一致，不引入回归；skip-topk 掩码作为 DSATokenToKVPool 新参数，默认全 False，非 DSA 池不受影响。
- 团队侧：indexer 内存预算计算收敛到 _compute_dsa_indexer_cell_size 单一入口，后续调整 DSA 内存模型只需改一处；测试新增 5 个策略用例把 HiCache/PD/draft 三个边界钉死。
- 风险标记：核心路径变更，HiCache/hisparse 回退门控，CP 分片保守 +1 预算，draft 全额计费，0 行 buffer 潜在越界

关联脉络

- PR #30310 Increase the KV cache pool when using indexShare by 15%: 本 PR 标题即为 Reland #30310，并在其基础上叠加 skip-topk 层 indexer 省略；#30310 因 HiCache/IMA 问题被否，本 PR 用门控修复后重放。
- PR #31324 [AMD] [GLM5] Skip DSA decode indexer when kv_len <= index_topk (dense k-only fast path): 同属 DSA indexer 计算省略方向：31324 跳过的 decode 侧 dense k-only 快路径，本 PR 跳过的是 skip-topk 层的 index-K 存储与 Indexer 模块，互为补充。
- PR #35110 [Fix] Read the DSA prefill CP flag from the parallel config bag in bootstrap: DSA 在 AMD 侧的 CP prefill 修复，与本 PR 的 CP 层分片预算计算（get_glm_dsa_cp_layer_shard_info）共用同一 DSA CP 配置链路。
- PR #30519 [AMD] [GLM5] fp8 MLA absorbed bmm for GLM-5.2 on gfx950: 同为 GLM-5.2 在特定硬件（gfx950/GB300）上的 MLA/ 量化性能优化，与本 PR 同属 GLM-5 稀疏注意力性能演进线。