

PR #37748 完整报告

sgl-project/sglang

[CPU] Implement fused QK Norm and RoPE kernels

合并时间: 2026-09-15 14:17

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/37748>

执行摘要

- 一句话: 新增 CPU 融合 QK-Norm/RoPE 内核并接入共享路径
- 推荐动作: 值得精读, 尤其是 CPU 后端、Kernel 与模型接入方向的同学; 如果只关心 CUDA 性能可以略读, 但 `apply_qk_norm` 的改动属于共享路径, 仍建议确认自己维护的模型是否受影响。重点看三处设计决策: 一是 CPU 版为何选择「消费预计算 `cos_sin_cache`」而非像 CUDA 版重算 YaRN 参数, 这既是性能取舍也是能力边界; 二是 `can_use_fused_qk_norm_cpu` 的门控条件集合 (`eps` 相等、`stride(-1)==1`、`head_dim` 整除) 如何界定原地写回的合法输入; 三是 `apply_qk_norm_rope` 被重构为单一 `use_fused` 分支后, `_used_fused_qk_norm_rope_last_call`、`fused_set_kv_buffer` 等既有语义是否在 CPU 路径下仍然成立。后续若要扩展, 优先补 `partial rotary_dim` 与 YaRN 场景测试, 并把 `_match_cos_sin_cache_dtype` 这类私有接口的使用显式化。

功能与动机

PR body 明确动机是「cut redundant kernel-launch/dispatch overhead for models (e.g. Qwen3-MoE) that apply RMSNorm to Q/K per head before RoPE」, 即 CPU 后端此前只能走「`apply_qk_norm + rotary_emb()`」两步路径, 存在额外的算子调度与中间张量开销; CUDA 侧早已有对应融合实现 (`qknorm.cuh`、`fused_qknorm_rope_kernel.cu`), CPU 侧缺失导致同一模型在不同后端性能不对称。PR body 还指出原实现存在「per-element scalar sin/cos calls」和「a per-call heap allocation」, 因此新内核直接按 `position_ids` 索引预计算 `cache` 来消除这两处开销。

实现拆解

1. 内核层新增两个融合实现 (`python/sglang/kernels/aot/csrc/cpu/norm.cpp`, +277): 在匿名命名空间中加入 `fused_qk_norm_per_head` (用 `at::vec::Vectorized` 以 float 累加 `sum(x^2)`, `#pragma GCC unroll 4` 展开, 尾部保留标量兜底)、`fused_qk_norm_kernel_impl` (以 `token * (num_q_heads + num_kv_heads) + head` 展平工作项交给 `at::parallel_for`, 用 `stride(0)` 计算 token 起始地址, 因此 qkv 切分出的非连续视图可直接原地改写)、`fused_qk_norm_rope_apply_interleaved` (用 `at::vec::deinterleave2 / interleave2` 成套处理 `(x, y)` 对) 与 `fused_qk_norm_rope_apply_neox` (前后半段分别加载), 最后由 `fused_qk_norm_rope_per_head` 按 `is_neox` 分派; 导出函数 `fused_qk_norm_cpu / fused_qk_norm_rope_cpu` 负责维度、dtype、`rotary_dim` 偶数与上界校验, 并把 `int32` 的

position_ids 一次性转成 int64 传给 kernel。

2. 算子注册层 (`python/sglang/kernels/aot/csrc/cpu/torch_extension_cpu.cpp`, +20) : 新增前向声明与 `TORCH_LIBRARY_FRAGMENT(ssl_kernel, m)` 中的 schema `fused_qk_norm_cpu(Tensor(a!) q, Tensor(b!) k, ...)`、`fused_qk_norm_rope_cpu(Tensor(a!) q, Tensor(b!) k, ..., bool is_neox, Tensor position_ids, Tensor cos_sin_cache, int rotary_dim)`，并用 `Tensor(a!) / Tensor(b!)` 显式标注原地语义，`m.impl(..., torch::kCPU, ...)` 绑定实现。
3. 共享路径接入 (`python/sglang/srt/models/utils.py`, +32/-1) : 新增 `@lru_cache(maxsize=1)` 的 `_has_cpu_fused_qk_norm()` 做一次 `hasattr(torch.ops.sgl_kernel, ...)` 探测，新增 `can_use_fused_qk_norm_cpu(q, k, head_dim, q_eps, k_eps)` 把资格收敛为「CPU + q/k eps 相同 + 2D + bf16/fp16 同 dtype + 最后一维 stride 为 1 + 最后一维能被 head_dim 整除 + 算子可用」，并在 `apply_qk_norm()` 的最前面插入该快路径；由于 `apply_qk_norm` 被大量模型共用，这一步决定了改动的影响半径。
4. Qwen3-MoE 的 Norm+RoPE 融合接入 (`python/sglang/srt/models/qwen3_moe.py`, +61/-24) : 新增 `_has_cpu_fused_qk_norm_rope()` 与 `Qwen3MoeAttention.use_fused_qk_norm_rope_cpu` (条件为 `_is_cpu`、非 `MRotaryEmbedding`、`rotary_dim % 2 == 0`、算子可用)，并把 `apply_qk_norm_rope` 从「只有 CUDA 一条 `use_fused`」重构为「统一 `use_fused` 判定 + 内层 `if _is_cuda: ... elif _is_cpu: ...`」；CPU 分支先 `qkv.split`，再调用 `self.rotary_emb._match_cos_sin_cache_dtype(q)` 对齐 dtype，最后以 `positions.view(-1)`、`cos_sin_cache`、`rotary_dim` 调算子，`_used_fused_qk_norm_rope_last_call` 在两条分支都置位；不满足条件时仍回退到 `apply_qk_norm + rotary_emb()` 并保留 `fused_set_kv_buffer_arg` 逻辑。
5. 测试配套 (`test/registered/cpu/test_norm.py`, +127) : 新增 `TestFusedQKNorm`，用 `_norm_per_head_native` 与手写逐元素 `apply_rope` 作为参考，覆盖 `test_fused_qk_norm` (bf16/fp16, 4 组形状含 `head_dim=96`，并额外断言 V 未被原地写污染) 与 `test_fused_qk_norm_rope` (bf16, interleaved/NeoX 双风格，手工构造 `cos_sin_cache` 并对比参考实现)。

关键文件：

- `python/sglang/kernels/aot/csrc/cpu/norm.cpp` (模块 融合算子；类别 source；类型 core-logic；符号 `fused_qk_norm_cpu`, `fused_qk_norm_rope_cpu`, `fused_qk_norm_per_head`, `fused_qk_norm_kernel_impl`) : 本 PR 的核心：新增 per-head RMSNorm 与 Norm+RoPE 融合的 C++ 实现，包含向量化 interleaved/NeoX 旋转、parallel_for 工作项划分、stride 感知的原地写回，以及全部输入校验逻辑。
- `python/sglang/srt/models/qwen3_moe.py` (模块 模型接入；类别 source；类型 core-logic；符号 `_has_cpu_fused_qk_norm_rope`, `Qwen3MoeAttention.init`, `Qwen3MoeAttention.apply_qk_norm_rope`) : Qwen3-MoE 的 Norm+RoPE 融合接入点：新增 CPU 资格判定，并把 `apply_qk_norm_rope` 重构为 CUDA/CPU 共用 `use_fused` 分支，是本次改动最复杂的控制流重构。

- python/sglang/srt/models/utils.py (模块 共享路径; 类别 source; 类型 core-logic; 符号 `_has_cpu_fused_qk_norm`, `can_use_fused_qk_norm_cpu`, `apply_qk_norm`) : 共享 `apply_qk_norm` 入口的改动: 新增 CPU 快路径与资格判定, 影响所有使用 QK-Norm 的模型, 是本次改动影响半径最大的文件。
- python/sglang/kernels/aot/csrc/cpu/torch_extension_cpu.cpp (模块 算子注册; 类别 source; 类型 data-contract; 符号 `fused_qk_norm_cpu`, `fused_qk_norm_rope_cpu`) : 算子注册层: 定义两个新 op 的 schema (含 `Tensor(a!)/Tensor(b!)` 原地标记) 并绑定 CPU 实现, 是新算子的对外接口契约。
- test/registered/cpu/test_norm.py (模块 单元测试; 类别 test; 类型 test-coverage; 符号 `TestFusedQKNorm`, `_norm_per_head_native`, `test_fused_qk_norm`, `test_fused_qk_norm_rope`) : 新增 `TestFusedQKNorm` 覆盖两个新算子, 包含手工 `cos_sin_cache` 构造与逐元素 RoPE 参考实现, 是本次正确性验证的主要依据。

关键符号: `fused_qk_norm_cpu`, `fused_qk_norm_rope_cpu`, `fused_qk_norm_per_head`, `fused_qk_norm_kernel_impl`, `fused_qk_norm_rope_kernel_impl`, `fused_qk_norm_rope_apply_interleaved`, `fused_qk_norm_rope_apply_neox`, `can_use_fused_qk_norm_cpu`, `_has_cpu_fused_qk_norm`, `_has_cpu_fused_qk_norm_rope`, `apply_qk_norm`, `Qwen3MoeAttention.apply_qk_norm_rope`, `TestFusedQKNorm.test_fused_qk_norm`, `TestFusedQKNorm.test_fused_qk_norm_rope`

关键源码片段

python/sglang/srt/models/qwen3_moe.py

Qwen3-MoE 的 Norm+RoPE 融合接入点: 新增 CPU 资格判定, 并把 `apply_qk_norm_rope` 重构为 CUDA/CPU 共用 `use_fused` 分支, 是本次改动最复杂的控制流重构。

```
@lru_cache(maxsize=1)
def _has_cpu_fused_qk_norm_rope() -> bool:
    # 只在首次调用时探测 CPU 扩展是否注册了该算子, 避免反复做 hasattr
    return hasattr(torch.ops.sgl_kernel, "fused_qk_norm_rope_cpu")

class Qwen3MoeAttention(nn.Module):
    def __init__(self, ...):
        ...
        # 资格判定在 __init__ 阶段做完: CPU、非 MRotaryEmbedding、rotary_dim
        # 为偶数、算子可用
        self.use_fused_qk_norm_rope_cpu = (
            _is_cpu
            and not isinstance(self.rotary_emb, MRotaryEmbedding)
            and self.rotary_emb.rotary_dim % 2 == 0
            and _has_cpu_fused_qk_norm_rope()
        )
        self._used_fused_qk_norm_rope_last_call = False

    def apply_qk_norm_rope(self, qkv, positions, forward_batch):
```

```

# CUDA 与 CPU 共用同一个 use_fused 判定，内部再按后端分流，避免两条独立分支漂移
use_fused = (self.use_fused_qk_norm_rope and qkv.dtype == torch.bfloat16) or (
    self.use_fused_qk_norm_rope_cpu
    and qkv.dtype in (torch.bfloat16, torch.float16)
)
if use_fused:
    if _is_cuda:
        # CUDA 版仍按原路径传入 YaRN 参数，每次前向重新计算 factor/low/high/attention_factor
        theta = self.rope_theta
        positions = (
            positions.view(-1).to(dtype=torch.int32, device=qkv.device).contiguous()
        )
        factor, low, high, attention_factor = compute_yarn_parameters(self.config)
        fused_qk_norm_rope(
            qkv, self.num_heads, self.num_kv_heads, self.num_kv_heads,
            self.head_dim, self.q_norm.variance_epsilon,
            self.q_norm.weight, self.k_norm.weight,
            theta, self.rotary_emb.is_neox_style, positions,
            factor, low, high, attention_factor,
        )
        q, k, v = qkv.split([self.q_size, self.kv_size, self.kv_size], dim=-1)
    elif _is_cpu:
        # CPU 版直接把 qkv 切成 q/k/v 视图，算子按 stride(0) 原地写回
        q, k, v = qkv.split([self.q_size, self.kv_size, self.kv_size], dim=-1)
        # 先把模块预计算的 cos_sin_cache 对齐到 q 的 dtype，C++ 侧强校验两者同 dtype
        self.rotary_emb._match_cos_sin_cache_dtype(q)
        torch.ops.sgl_kernel.fused_qk_norm_rope_cpu(
            q,
            k,
            self.q_norm.weight,
            self.k_norm.weight,
            self.q_norm.variance_epsilon,
            self.rotary_emb.is_neox_style,
            positions.view(-1),
            # 复用初始化时构建一次的缓存，省掉每 token 的 sin/cos 计算与堆分配
            self.rotary_emb.cos_sin_cache,
            self.rotary_emb.rotary_dim,
        )
        self._used_fused_qk_norm_rope_last_call = True
    else:
        # 回退到非融合实现：apply_qk_norm + rotary_emb，并保留 fused_set_kv_buffer 优化
        q, k, v = qkv.split([self.q_size, self.kv_size, self.kv_size], dim=-1)
        q, k = apply_qk_norm(
            q=q, k=k, q_norm=self.q_norm, k_norm=self.k_norm,
            head_dim=self.head_dim, alt_stream=self.alt_stream,
        )
        q, k = self.rotary_emb(
            positions, q, k,

```

```

        fused_set_kv_buffer_arg=(
            create_fused_set_kv_buffer_arg(
                value=v, layer=self.attn, forward_batch=forward_batch,
            )
            if enable_fused_set_kv_buffer(forward_batch)
            and self.compatible_with_fused_kv_buffer
            else None
        ),
    )
    self._used_fused_qk_norm_rope_last_call = False
    return q, k, v

```

python/sglang/srt/models/utils.py

共享 apply_qk_norm 入口的改动：新增 CPU 快路径与资格判定，影响所有使用 QK-Norm 的模型，是本次改动影响半径最大的文件。

```

@lru_cache(maxsize=1)
def _has_cpu_fused_qk_norm() -> bool:
    # 探测 CPU 扩展是否导出 fused_qk_norm_cpu, 结果只缓存一次
    return hasattr(torch.ops.sgl_kernel, "fused_qk_norm_cpu")

```

```

def can_use_fused_qk_norm_cpu(
    q: torch.Tensor, k: torch.Tensor, head_dim: int, q_eps: float, k_eps: float
) -> bool:
    return (
        _is_cpu
        and q_eps == k_eps # 融合算子在 q/k 之间只接受一个 eps
        and q.dim() == 2
        and k.dim() == 2
        and q.dtype in (torch.bfloat16, torch.float16)
        and k.dtype == q.dtype
        # q/k 通常是 qkv 切出的 strided view; 算子只要求 head 内维度连续
        and q.stride(-1) == 1
        and k.stride(-1) == 1
        and q.size(-1) % head_dim == 0
        and k.size(-1) % head_dim == 0
        and _has_cpu_fused_qk_norm()
    )

```

```

def apply_qk_norm(
    q: torch.Tensor,
    k: torch.Tensor,
    q_norm: RMSNorm,
    k_norm: RMSNorm,
    head_dim: int,
    alt_stream: Optional[torch.cuda.Stream] = None,
    allow_inplace: bool = True,

```

```

) -> Tuple[torch.Tensor, torch.Tensor]:
    ...
    batch_size = q.size(0)
    q_eps = q_norm.variance_epsilon
    k_eps = k_norm.variance_epsilon

    # CPU 快路径放在最前面：原地写回，省掉 reshape 与两次独立 RMSNorm 调用
    if allow_inplace and can_use_fused_qk_norm_cpu(q, k, head_dim, q_eps, k_eps):
        torch.ops.sgl_kernel.fused_qk_norm_cpu(
            q, k, q_norm.weight, k_norm.weight, q_eps
        )
        return q, k

    # 以下 CUDA / ROCm / reshape 分支保持原有逻辑不变
    if (
        _is_cuda # TODO(dark): have not tested on ROCm or other backends
        and allow_inplace # TODO(dark): this can be relaxed if needed
        and (q_eps == k_eps) # TODO(dark): this can also be relaxed
        and not envs.SGLANG_ENABLE_DETERMINISTIC_INFERENCE.get()
        and get_exec().graph.cuda_graph_config.prefill.tc_compiler != "inductor"
        and can_use_fused_inplace_qk_norm(head_dim, q.dtype)
    ):
        fused_inplace_qk_norm(
            q=q.view(batch_size, -1, head_dim),
            k=k.view(batch_size, -1, head_dim),
            q_weight=q_norm.weight,
            k_weight=k_norm.weight,
            head_dim=head_dim,
            eps=q_eps,
        )
        return q, k

```

test/registered/cpu/test_norm.py

新增 TestFusedQKNorm 覆盖两个新算子，包含手工 cos_sin_cache 构造与逐元素 RoPE 参考实现，是本次正确性验证的主要依据。

```

class TestFusedQKNorm:
    def _norm_per_head_native(
        self, x: torch.Tensor, weight: torch.Tensor, head_dim: int, eps: float
    ):
        # 参考实现：reshape 到 (... , head_dim) 后 float 计算 rsqrt(mean(x^2) + eps) 再乘权重
        out = x.reshape(-1, head_dim).float()
        out = out * torch.rsqrt(out.pow(2).mean(-1, keepdim=True) + eps)
        out = out.to(x.dtype) * weight
        return out.reshape(x.shape)

    @pytest.mark.parametrize("dtype", DTYPES, ids=DTYPE_IDS)
    @pytest.mark.parametrize(
        "batch_size,num_head,num_head_kv,head_dim",

```

```

    [(1, 16, 2, 128), (9, 8, 1, 64), (256, 16, 2, 128), (4109, 3, 1, 96)],
)
def test_fused_qk_norm(self, batch_size, num_head, num_head_kv, head_dim, dtype):
    q_size = num_head * head_dim
    kv_size = num_head_kv * head_dim
    qkv = torch.randn([batch_size, q_size + 2 * kv_size], dtype=dtype)
    # q/k 是 qkv 的 strided view, 与模型里的真实调用形态一致
    q, k, v = qkv.split([q_size, kv_size, kv_size], dim=-1)
    v_before = v.clone()

    ref_q_out = self._norm_per_head_native(q, q_weight, head_dim, eps)
    ref_k_out = self._norm_per_head_native(k, k_weight, head_dim, eps)

    # 原地调用: q/k 被直接改写
    torch.ops.sgl_kernel.fused_qk_norm_cpu(q, k, q_weight, k_weight, eps)

    torch.testing.assert_close(q, ref_q_out, atol=atol, rtol=rtol)
    torch.testing.assert_close(k, ref_k_out, atol=atol, rtol=rtol)
    # 原地写回不得越过 q/k 的 head 行污染 V, 这里要求零误差
    torch.testing.assert_close(v, v_before, atol=0, rtol=0)

def test_fused_qk_norm_rope(self, dtype, is_neox, num_head, num_head_kv, head_dim):
    ...
    # 用纯 Python 逐元素实现生成参考结果, NeoX 与 interleaved 两套布局分别验证
    def apply_rope(x: torch.Tensor, pos: int) -> torch.Tensor:
        ...

    # 按算子约定手工构造 cos_sin_cache: 前半 cos、后半 sin, 形状 (num_tokens, rotary_dim)
    half = head_dim // 2
    freqs = base ** (-2.0 * torch.arange(half, dtype=torch.float32) / head_dim)
    theta = position_ids.to(torch.float32)[: , None] * freqs[None, :]
    cos_sin_cache = torch.cat([torch.cos(theta), torch.sin(theta)], dim=-1).to(dtype)

    torch.ops.sgl_kernel.fused_qk_norm_rope_cpu(
        q, k, q_weight, k_weight, eps, is_neox, position_ids, cos_sin_cache, head_dim
    )

    torch.testing.assert_close(q, ref_q, atol=atol, rtol=rtol)
    torch.testing.assert_close(k, ref_k, atol=atol, rtol=rtol)

```

评论区精华

本 PR 的评审记录非常薄: 唯一的 review 是 mingfeima 的空 body **APPROVED**, 唯一的 issue 评论是作者 cyxlily 自己发的 [/tag-run-ci-label](#), 没有任何 inline review 评论。因此没有形成显式的技术交锋, 以下几点是从实现差异中能读出的「本可讨论但未讨论」的议题: 其一, CPU 算子签名刻意不接收 **base/factor/low/high/attention_factor**, 而是直接消费模块层预计算的 **cos_sin_cache** 与 **rotary_dim**, 这与 CUDA 版每次调用重算 YaRN 参数的做法形成对照, 属于隐式能力差异; 其二, **apply_qk_norm** 作为跨模型共享入口被插入了 CPU 快路径,

但资格判定完全依赖形状与 dtype 门控，review 未讨论其潜在外溢；其三，PR body 中记录的 Base / Extra / AMD ROCm 三组 CI 状态均为失败标记，最终仍以一次 approve 合入，PR 内没有留下失败原因是否与本次 CPU 改动相关的结论。

- CPU 版算子为何不接收 YaRN 参数而是直接索引 `cos_sin_cache` (design): 实现选择复用 `rotary_emb` 的预计算缓存以消除每 token 的 `sin/cos` 计算与堆分配，代价是判定条件变弱、能力边界需要靠缓存内容保证；review 未对该取舍表态。
- 共享 `apply_qk_norm` 中插入 CPU 快路径的外溢范围 (correctness): 测试通过 `assert_close(v, v_before, atol=0, rtol=0)` 覆盖了「原地写不越界污染 V」，但更广义的调用方兼容性没有讨论结论；实际合入依赖 `can_use_fused_qk_norm_cpu` 的门控足够严格。
- CI 状态与合入依据 (testing): 维护者以一次 approve 接受了该变更，但 PR 记录中没有说明 CI 红灯是否与本次 CPU 改动相关，也没有精度 / 性能回归的独立复现说明。

风险与影响

• 风险:

1. 共享路径回归: `apply_qk_norm` (`python/sglang/srt/models/utils.py`) 是所有使用 QK-Norm 模型的公共入口，本次在其最前面新增了原地写回分支。任何满足 `can_use_fused_qk_norm_cpu` 形状门控的调用都会从「reshape + RMSNorm 模块」切换为「C++ 原地改写」，若某模型后续依赖 q/k 的非原地语义或依赖 q/k 的 stride 布局，会出现难以定位的静默错误；测试里 `assert_close(v, v_before, atol=0, rtol=0)` 只覆盖了「不越界写 V」这一种越界形态。
2. RoPE 语义等价性缺口: CPU 算子不含 YaRN 参数路径，资格判定只排除了 `MRotaryEmbedding` 并检查 `rotary_dim % 2 == 0`，没有像 CUDA 分支那样检查 `_yarn_factor != 1.0`。等价性完全依赖 `self.rotary_emb.cos_sin_cache` 已把 `scaling/mscale` 烘焙进缓存；若某模型的 RoPE 模块在 forward 阶段另有后处理（例如额外的乘子或动态缩放），CPU 融合路径会与其 fallback 不等价，建议针对 YaRN 模型做一次端到端精度验证。
3. 私有 API 耦合与状态突变: CPU 分支调用 `self.rotary_emb._match_cos_sin_cache_dtype(q)` 修改 RoPE 模块内部缓存 dtype，且 C++ 侧强校验 `cos_sin_cache.scalar_type() == q.scalar_type()`。一旦回退路径与融合路径交替触发，缓存的 dtype 状态会在两种精度间切换，属于跨层耦合带来的隐性风险。
4. 覆盖缺口: `test_fused_qk_norm_rope` 只测 bf16、`num_head_kv` 固定为 2 与 1，未覆盖 fp16、partial rotary (`rotary_dim < head_dim`，而算子签名与 `TORCH_CHECK(rotary_dim <= head_dim)` 明确支持该场景)、以及非 `MRotaryEmbedding` 的 `scaling` 场景；`head_dim` 非向量宽度整数倍时的尾部标量循环也缺少显式用例。
5. 收益不均衡与 CI 疑点: PR 自带数据显示 batch=1024、bf16 时 norm 融合仅 1.019x（几乎无收益），说明大 batch 下收益有限、收益主要来自 RoPE 融合；同时 PR body 记录的 CI (Run #34449099991、#34561508953 等) 均为失败状态，合入前缺少公开结论。
- 影响: 对用户: 在 Intel x86 (AVX512 等) CPU 部署上运行 Qwen3-MoE 等「per-head QK-Norm + RoPE」模型的用户，可获得 QK-Norm 阶段约 1.0–1.7x、QK-Norm+RoPE 阶段约 1.85–4.33x 的局部加速，长序列小 batch 场景收益更明显；由

于算子可用性与 `_is_cpu` 双重门控，未编译新算子的环境会自动回退到原路径，无行为破坏。对系统：sgl-kernel CPU AOT 扩展新增两个对外 op schema（原地语义），成为公开接口面，需要与 CUDA 版 `fused_qknorm_rope` 保持长期语义对齐；`apply_qk_norm` 与 `Qwen3MoeAttention.apply_qk_norm_rope` 的分支结构变复杂，后续改动需同时考虑 CUDA 与 CPU 两条融合路径。对团队：这是 Intel CPU 后端继 `fused_qk_gemma_rmsnorm_with_gate_cpu` 之后在 QK-Norm 家族上的继续补齐，形成了与 CUDA 侧对称的融合算子布局，也为后续把该快路径推广到更多模型（只要它们走 `apply_qk_norm`）铺平了路。- 风险标记：核心推理路径变更，共享路径改动，CPU 专属优化，缺少测试覆盖，RoPE 能力差异，私有 API 耦合，CI 状态未闭环

关联脉络

- PR #38687 [Kernel] Add OOT dispatch for clamp position: 同属「内核实现 + 模型侧 OOT 分发入口 + 注册 kernels 单测」的演进线，两者都是把原本分散的算子调用收敛为一个可探测、可回退的分发入口。
- PR #35233 [AMD] Fix registered HiCache host pointer aliases: 同样在 sgl-kernel AOT/JIT 扩展层新增或修正 CPU/ 异构侧算子（kvcacheio）的指针与 dtype 语义，并配套 kernels 单测，是同类扩展接口的近期参照。
- PR #36821 [KDA] Support ReplaySSM ring-write in the fused chain-verify kernel: 同为「融合 kernel 以削减 launch 与重复访存开销」的性能类改动，采用 backend 资格判定 + 单测 + benchmark 的同一套落地范式，可作为评估本 PR 收益方法的对照。