

PR #53566 完整报告

vllm-project/vllm

[5/N][warmup][DSv4] Migrate NVIDIA CuTeDSL attention kernels

合并时间: 2026-09-12 12:06

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/53566>

执行摘要

- 一句话: DSv4 CuTeDSL 注意力内核迁至共享预热契约
- 推荐动作: 值得精读。这个 PR 是理解 vLLM 「零运行时 JIT」基础设施的关键样本: `kernel_launcher` 的通用化设计、CuTeDSL fake-tensor compile-only warmup、`CompileKey` 与 AST 追踪 dispatch 的结合方式, 都值得后续内核迁移直接复用。重点关注两点: 一是 warmup key 与运行时 dispatch 的一致性校验方法 (FP32 RoPE cache 的教训), 二是跨平台 gate 的写法 (CUDA 专属内核注册必须与运行时平台条件完全镜像)。也可以对照 Issue #49349 的跟踪列表, 评估本 PR 与 #56323、#50109 的衔接。

功能与动机

关联 Issue #49349 明确提出「Zero JIT compilation during runtime」: 通过共享 JIT warmup 基础设施 (#47456 提出) 让所有内核在启动预热阶段完成编译, 最终目标是模型能在 `--jit-monitor-mode error` 下无任何运行时 JIT 编译地运行。PR body 说明本 PR 将 DSv4 的 NVIDIA CuTeDSL attention 与 sparse-attention 内核迁移到共享 warmup 契约, 定义 kernel-owned compile keys 与 AST-traced warmup spaces, 并加入 compile-only CuTeDSL 路径 (fake tensor descriptors), 同时保留既有内核体、运行时 dispatch 与 specialization 行为。

实现拆解

整个迁移分四步完成:

1. 通用 launcher 契约收敛: 在 `vllm/model_executor/warmup/jit_warmup.py` 新增通用 `kernel_launcher` 装饰器, 并在 `VllmJitKernel` 基类上新增 `bind_launch_inputs` 类属性 (默认 True)。原 Triton 专属的 `kernel_launcher` (`jit_warmup_triton_helper.py`) 被删除, `VllmTritonJitKernel.launch` 的签名改为接收完整 `launch_spec` (grid + launch_kwargs + outputs), 解包逻辑从装饰器下沉到 launch 方法; CuTeDSL 侧同样删除专用 `cutedsl_kernel_launcher`, 改为在 `VllmCuTeDSLJitKernel` 上实现 `launch(launch_spec, _inputs)`, 并声明 `bind_launch_inputs = False` 以跳过参数名绑定。
2. CuTeDSL launch spec 扩展: `CuTeDSLLaunchSpec` 从固定的二元组扩展为 2/3/4 元组联合类型, 分别表示 (compile_key, args)、额外携带输出占位、以及携带 post-launch finalizer 回调; `launch` 方法按 spec 长度分别返回 executor 结果、声明输出或 finalizer 返回值。同时在 `vllm/cute_utils/init.py` 新增 `torch_to_cute_dtype`, 替换各内核文件里重复的私有 dtype 映射字典。

3. DSv4 NVIDIA 内核逐个迁移：vllm/models/deepseek_v4/nvidia/ops/fused_indexer_q_cutedsl.py 将原 `IndexerQRopeQuantKernel` 基类拆分为 `IndexerQMxFp4Kernel` / `IndexerQFp8Kernel`，均继承 `VllmCuTeDSLJitKernel`，内部定义 frozen `CompileKey`（`head_dim`、`rope_dim`、`num_heads`、`cos_sin_dtype`、`coarsen`）、`kernel` 静态方法、`dispatch`、`warmup_inputs` 与 `get_warmup_keys`，`__call__` 用 `@kernel_launcher` 装饰；`dequant_gather_k_cutedsl.py` 的 `DequantGatherKCacheKernel` 同样改为继承 `VllmCuTeDSLJitKernel`，`CompileKey` 由（`block_size`，`has_gather_lens`）组成，原 `functools.cache` 式每次编译改为 `warmup` 缓存命中；`sparse_attn_compress_cutedsl.py` 中各压缩内核（C128 block8、C4 存储、full-store 等）改为模块级单例 `_SPARSE_ATTN_*_KERNEL` 并各自注册 `warmup`。
4. 模型层接线与 `warmup` 注册：`compressor.py` 在 `DeepseekCompressor.__init__` 中按 `current_platform.is_cuda()` and `head_dim == 512` 分支注册 CuTeDSL 压缩内核，`head_dim` 较大的 Triton 路径保留原分支；`attention.py` 的 `DeepseekV4Indexer` 在 `has_cutedsl()` 且 CUDA 时注册 `_INDEXER_Q_FP8/MXFP4_KERNEL`，并在 FLASHMLA_SPARSE_DSV4 后端下按 `cutedsl` 可用性分别注册 CuTeDSL 或 Triton 的 `dequant-gather` 内核。测试配套包括新增 `tests/models/test_deepseek_v4_nvidia_attention_jit_warmup.py` 验证 `warmup key` 与运行时 `dispatch` 的一致性，以及更新 `tests/model_executor/` 下两个 `launcher` 测试（新增 `finalizer` 场景）。

关键文件：

- `vllm/model_executor/warmup/jit_warmup_cutedsl_helper.py`（模块 预热框架；类别 `source`；类型 `data-contract`；符号 `cutedsl_kernel_launcher`，`wrapper`，`launch`）：CuTeDSL 预热契约的核心改造点：删除专用 `cutedsl_kernel_launcher`，`CuTeDSLLaunchSpec` 扩展为 2/3/4 元组，`VllmCuTeDSLJitKernel` 新增 `bind_launch_inputs=False` 与 `launch` 方法，`warmup` 与运行时共享编译缓存。
- `vllm/model_executor/warmup/jit_warmup.py`（模块 预热框架；类别 `source`；类型 `data-contract`；符号 `kernel_launcher`，`wrapper`）：新增通用 `kernel_launcher` 装饰器与 `VllmJitKernel.bind_launch_inputs` 属性，是整个 Triton/CuTeDSL `launcher` 收敛的基座。
- `vllm/model_executor/warmup/jit_warmup_triton_helper.py`（模块 预热框架；类别 `source`；类型 `data-contract`；符号 `kernel_launcher`，`wrapper`）：Triton 侧 `launcher` 下放：删除本地 `kernel_launcher` 装饰器，`launch` 改收完整 `launch_spec`，统一返回 `outputs` 语义。
- `vllm/models/deepseek_v4/nvidia/ops/fused_indexer_q_cutedsl.py`（模块 索引器内核；类别 `infra`；类型 `infrastructure`；符号 `fused_indexer_q_rope_quant_mxfp4_cutedsl`，`fused_indexer_q_rope_quant_fp8_cutedsl`，`_load_q_and_rope`，`IndexerQRopeQuantKernel`）：Indexer-Q 融合内核（MXFP4/FP8 两个变体）从基于 `functools.cache` 的即时编译迁移到 `VllmCuTeDSLJitKernel` + `CompileKey` + `kernel_launcher` 契约，改动量最大（+591/-510）。
- `vllm/models/deepseek_v4/nvidia/ops/dequant_gather_k_cutedsl.py`（模块 去量化收集；类别 `infra`；类型 `infrastructure`；符号 `dequantize_and_gather_k_cache_cutedsl`，`DequantGatherKCacheKernel`，`init`，`call`）：`dequantize-and-gather K cache` 内核迁移为 `VllmCuTeDSLJitKernel` 子类，`CompileKey` =（`block_size`，`has_gather_lens`），原每次调用

即时编译改为预热缓存。

- `vllm/models/deepseek_v4/nvidia/ops/sparse_attn_compress_cutedsl.py` (模块 稀疏压缩; 类别 `infra`; 类型 `infrastructure`): 稀疏注意力压缩内核家族 (C128 block8、C4 存储、full-store 等) 全部改为模块级单例并注册 `warmup`, 是 diff 量最大的文件 (+1945/-1606), 也是 review 中 `blocking` 问题 (`wrapper` 删除导致测试 `ImportError`) 的发生地。
- `vllm/models/deepseek_v4/compressor.py` (模块 压缩器; 类别 `source`; 类型 `data-contract`): `DeepseekCompressor` 的 `warmup` 注册按 `CUDA + head_dim == 512` 分支选择 `CuTeDSL` 压缩内核, 修复了 `LucasWilkinson P1` 指出的 `ROCm/XPU` 误注册 `NVIDIA` 内核问题。
- `vllm/models/deepseek_v4/attention.py` (模块 注意力层; 类别 `source`; 类型 `data-contract`): `DeepseekV4Indexer` 增加 `indexer-Q` 的 `CuTeDSL` `warmup` 注册, 并按 `has_cutedsl()` 分流 `dequant-gather` `warmup`, 接好新内核单例。
- `vllm/model_executor/kernels/attention/dsa/dcp_indexer_cutedsl.py` (模块 索引器内核; 类别 `source`; 类型 `data-contract`): `DCP top-k` 候选打包内核同步迁移到通用 `kernel_launcher`, 去掉旧的专用装饰器与 `runtime_context` 传参。
- `vllm/cute_utils/__init__.py` (模块 类型转换; 类别 `source`; 类型 `core-logic`; 符号 `torch_to_cute_dtype`): 新增 `torch_to_cute_dtype` 公共辅助, 消除多个 `CuTeDSL` 内核文件里重复的 `torch dtype` 映射。
- `tests/model_executor/test_jit_warmup_cutedsl_launcher.py` (模块 预热测试; 类别 `test`; 类型 `test-coverage`; 符号 `test_cutedsl_launcher_runs_post_launch_finalizer`, `compile_cutedsl`, `executor`, `FinalizedKernel`): 测试从专用装饰器切换到通用 `kernel_launcher`, 并新增 `post-launch finalizer` 场景, 验证四元组 `launch spec` 的行为。
- `tests/model_executor/test_jit_warmup_triton_launcher.py` (模块 预热测试; 类别 `test`; 类型 `test-coverage`; 符号 `launch`): `Triton launcher` 测试随 `launch` 签名变化同步调整, 保证 `Wrapper` 到 `launch` 的参数传递语义不回退。

关键符号: `kernel_launcher`, `VllmCuTeDSLJitKernel.launch`, `VllmTritonJitKernel.launch`, `IndexerQMxFp4Kernel`, `IndexerQFp8Kernel`, `DequantGatherKCacheKernel`, `_load_q_and_rope`, `torch_to_cute_dtype`, `DeepseekCompressor.init(warmup 分支)`, `DeepseekV4Indexer.init(warmup 分支)`

关键源码片段

`vllm/model_executor/warmup/jit_warmup_cutedsl_helper.py`

`CuTeDSL` 预热契约的核心改造点: 删除专用 `cutedsl_kernel_launcher`, `CuTeDSLLaunchSpec` 扩展为 2/3/4 元组, `VllmCuTeDSLJitKernel` 新增 `bind_launch_inputs=False` 与 `launch` 方法, `warmup` 与运行时共享编译缓存。

```
class VllmCuTeDSLJitKernel(VllmJitKernel[CompileKeyT], Generic[CompileKeyT]):
    """CuTeDSL 内核所有者: warmup 与运行时共享同一份编译缓存。"""

    kernel: ClassVar[Any]
    # CuTeDSL 的 launch 参数已编码在 cutlass Tensor 视图中,
    # 通用 binder 不需要按参数名回填, 因此关闭 bind_launch_inputs。
```

```
bind_launch_inputs = False
```

```
@abstractmethod
```

```
def warmup_inputs(self, compile_key: CompileKeyT) -> tuple[Any, ...]:
```

```
    """返回用于编译一个 executor 特化版本的 fake 参数。"""
```

```
    raise NotImplementedError
```

```
def compile(self, compile_key: CompileKeyT) -> None:
```

```
    if compile_key in self._compiled_cache:
```

```
        return
```

```
    # 使用 fake stream 与 fake tensor 描述符编译入口,
```

```
    # 产物缓存在 _compiled_cache, 运行时直接复用。
```

```
    self._compiled_cache[compile_key] = compile_cutedsl(
```

```
        self.kernel(compile_key),
```

```
        *self.warmup_inputs(compile_key),
```

```
    )
```

```
def launch(
```

```
    self,
```

```
    launch_spec: CuTeDSLLaunchSpec[CompileKeyT],
```

```
    _inputs: Mapping[str, Any],
```

```
) -> Any:
```

```
    # launch_spec 结构: (compile_key, args), 可选附带输出占位与 epilogue。
```

```
    compile_key, launch_args = launch_spec[:2]
```

```
    # 未命中缓存时会在监控下编译, 运行时 JIT 会被 jit-monitor 捕获。
```

```
    executor = self._get_or_compile(compile_key)
```

```
    result = executor(*launch_args)
```

```
    if len(launch_spec) == 4:
```

```
        # 四元组: 执行 finalizer 返回最终输出,
```

```
        # 用于 " 写缓存后再读取 " 的封装场景。
```

```
        return launch_spec[3]()
```

```
    # 三元组: 返回声明的输出占位; 二元组: 返回 executor 的结果。
```

```
    return launch_spec[2] if len(launch_spec) == 3 else result
```

vllm/model_executor/warmup/jit_warmup.py

新增通用 kernel_launcher 装饰器与 VllmJitKernel.bind_launch_inputs 属性, 是整个 Triton/CuTeDSL launcher 收敛的基座。

```
def kernel_launcher(call_fn: Callable[..., Any]) -> Callable[..., Any]:
```

```
    """将声明式 launch spec 委托给后端 owner 的通用装饰器。
```

```
    被装饰的 __call__ 只负责根据运行参数计算 compile key 与 launch 参数,
```

```
    实际的编译、缓存与启动统一由 self.launch(launch_spec, inputs) 完成。
```

```
    Triton 与 CuTeDSL 后端共用该装饰器, 通过 bind_launch_inputs 控制
```

```
    是否需要把调用参数按名字回填给内核。
```

```
    """
```

```
    signature = inspect.signature(call_fn)
```

```
@wraps(call_fn)
```

```

def wrapper(self: Any, *args: Any, **kwargs: Any) -> Any:
    launch_spec = call_fn(self, *args, **kwargs)
    if not self.bind_launch_inputs:
        # CuTeDSL 内核：参数已编码在 cutlass Tensor 视图中，无需绑定。
        return self.launch(launch_spec, {})
    # Triton 内核：把绑定后的调用参数回填给 launch，
    # 用于按名映射到内核形参或 <name>_ptr 指针参数。
    bound = signature.bind(self, *args, **kwargs)
    bound.apply_defaults()
    inputs = {
        name: value for name, value in bound.arguments.items() if name != "self"
    }
    return self.launch(launch_spec, inputs)

return wrapper

```

vllm/models/deepseek_v4/nvidia/ops/fused_indexer_q_cutedsl.py

Indexer-Q 融合内核（MXFP4/FP8 两个变体）从基于 `functools.cache` 的即时编译迁移到 `VllmCuTeDSLJitKernel+CompileKey+kernel_launcher` 契约，改动量最大（+591/-510）。

```

@cute.jit
def _load_q_and_rope(
    positions: cute.Tensor,
    q: cute.Tensor,
    cos_sin_cache: cute.Tensor,
    num_heads: cutlass.Constexpr,
    coarsen: cutlass.Constexpr,
    tb_size: cutlass.Constexpr,
    subwarp_size: cutlass.Constexpr,
    threads_per_token: cutlass.Constexpr,
    nope_dim: cutlass.Constexpr,
    rope_dim: cutlass.Constexpr,
    cos_sin_dtype: cutlass.Constexpr,
):
    """加载 Q（BF16）并对尾部的 rope_dim 应用交错 RoPE。

    返回 (q_bf16x2, tid, global_tid, sublane, token_id, head_tile_id,
        head_start, in_bounds, num_token_heads)，其中 q_bf16x2 是
    (coarsen, 8) 的 Uint32 rmem tile，每个线程覆盖 16 个 BF16 lane。
    """
    block_id, _, _ = cute.arch.block_idx()
    tid, _, _ = cute.arch.thread_idx()
    num_tokens = q.shape[0]
    num_token_heads = num_tokens * num_heads
    global_tid = block_id * tb_size + tid
    global_subwarp_id = global_tid // subwarp_size
    sublane = tid % subwarp_size
    token_id = global_subwarp_id // (num_heads // coarsen)
    head_tile_id = global_subwarp_id % (num_heads // coarsen)

```

```

head_start = head_tile_id * coarsen

# NOTE: token_id 可能越界，因此必须有 load/store guard;
# 不能提前 exit: CuTeDSL 不支持，且 warp shuffle 要求整 warp 活跃。
# 当线程块正好覆盖整数个 token 时 (threads_per_token % tb_size == 0) ,
# 编译器可以移除边界检查；否则 guard 必须保留。
must_in_bounds = cutlass.const_expr(threads_per_token % tb_size == 0)
in_bounds = must_in_bounds or (token_id < num_tokens)

cp_op = cute.nvgpu.CopyUniversalOp()
_layout = cute.make_layout((coarsen, 8), stride=(8, 1))
q_bf16x2 = cute.make_rmem_tensor(_layout, Uint32)

if in_bounds:
    # cute.copy 希望第一个 mode 被 copy atom 覆盖，因此先用 local_tile
    # 按 (coarsen, 16) 分块，再按 thread 的 sublane 取数据。
    q_tile = cute.local_tile(
        q[token_id, None, None],
        tiler=(coarsen, 16),
        coord=(head_tile_id, sublane),
    )
    cp_u32x8 = cute.make_copy_atom(cp_op, Uint32, num_bits_per_copy=256)
    for i in cutlass.range_constexpr(coarsen):
        src = cute.recast_tensor(q_tile[i, None], Uint32)
        cute.copy(cp_u32x8, src, q_bf16x2[i, None])

# RoPE 只作用于尾部 rope_dim 值；cos_sin_cache 布局为 [max_pos, rope_dim],
# 首 half 是 cos，次 half 是 sin。
if in_bounds and sublane * 16 >= nope_dim:
    cos_vals = cute.make_rmem_tensor((8,), Float32)
    sin_vals = cute.make_rmem_tensor((8,), Float32)
    pos = positions[token_id]
    cos_id = sublane - nope_dim // 16
    sin_id = cos_id + rope_dim // 16
    cos_src = cute.local_tile(cos_sin_cache[pos, None], tiler=(8,), coord=(cos_id,))
    sin_src = cute.local_tile(cos_sin_cache[pos, None], tiler=(8,), coord=(sin_id,))
    # ... (cos_sin_dtype 为 FP32 或 BF16 时分别加载并展开交错旋转，
    # 结果写回 q_bf16x2 的对应 lane)

return (
    q_bf16x2, tid, global_tid, sublane, token_id,
    head_tile_id, head_start, in_bounds, num_token_heads,
)

```

评论区精华

审阅中最重要的交锋集中在「warmup 键必须与运行时实际选择完全一致」这一主题：

- LucasWilkinson [P1]: compressor.py 中 head_dim == 512 的 warmup 注册没有平台过滤, ROCm/XPU 在开启 JIT warmup 时会尝试编译 NVIDIA-only 的 CuTe/nvgpu 内核, 而运行时 dispatch 只在 is_cuda() 下走 CuTeDSL。作者最终将条件改为 current_platform.is_cuda() and self.head_dim == 512, 与运行时逻辑对齐。
- LucasWilkinson (blocking): 删除 fused_kv_compress_norm_rope_insert_sparse_attn_cutedsl 导致 tests/kernels/test_compressor_kv_cache.py 的四个参数化用例全部 ImportError; 作者恢复了一个 thin wrapper 转发到新单例。
- LucasWilkinson P2: 指出 DSv4 的 DeepseekV4ScalingRotaryEmbedding 将 cos_sin_cache 保持在 FP32 (deepseek_scaling_rope.py), 因此按模型 dtype 生成 warmup key 会编译 BFloat16 特化、运行时却请求 Float32, 造成又一次 JIT 编译; 建议按实际 rope cache dtype 或直接用 Float32。作者回复称已改为「now uses only vllm_config.model_config.dtype」, 随后审阅者再次澄清 FP32 问题, 最终状态在 head 中未能完全确认。
- LucasWilkinson: 新 orchestrator 改为直接调用单例后丢失了 block_size == 8 的强校验, 而 SparseAttnCompressC128Block8Kernel 硬编码 state_block_size=8, 其他 block size 会静默读错行; 作者补上了防御性检查。
- LucasWilkinson: 运行时 store_full_fp8 由 kv_cache.dtype == torch.float8_e4m3fn 推导, fp8_e5m2/fp8_inc 等变体会 warmup 到 False 键而运行时请求 True 键; 作者说明 fp8_ds_mla 由独立内核覆盖 (C4/C128 各自注册)。
- coderabbitai: tests/models/test_deepseek_v4_nvidia_attention_jit_warmup.py 在非 SM100 CUDA 设备上运行 MXFP4 用例会因 dsa_indexer_uses_fp4() 抛 ValueError, 建议 mock 或 skip; 另有关于未使用的 VllmCuTeDSLJitKernel/ 专用 decorator 的清理建议 (最终以删除专用 decorator、统一通用 launcher 收尾)。
- compressor.py 的 CuTeDSL warmup 注册缺少 CUDA gate (correctness): 作者改为 if current_platform.is_cuda() and self.head_dim == 512, 与运行时平台条件镜像, 非 CUDA 平台继续走 Triton 注册。
- 删除 fused_kv_compress_norm_rope_insert_sparse_attn_cutedsl 导致测试 ImportError (testing): 作者恢复 thin wrapper 转发到新的 C4/full-store 单例, 保住既有测试入口。
- warmup 用模型 dtype 而非实际 FP32 RoPE cache (correctness): head 版本最终状态未完全确认; 作者在后续提交中调整了 warmup dtype 枚举, 但评审线程未完全闭环。
- C128 compress 路径丢失 block_size 校验 (correctness): 作者添加 if block_size != self.c128_compress.state_block_size 防御性检查。
- fp8 变体的 store_full_fp8 warmup 键与运行时不一致 (performance): 作者说明 fp8_ds_mla 由独立内核覆盖 (C4: SparseAttnCompressNormRopeStoreC4Kernel, C128: C128Block8 + NormRopeStoreKernel), 但非 fp8_ds_mla 的 fp8 变体覆盖情况未完全确认。
- MXFP4 warmup-key 测试在非 SM100 CUDA 设备失败 (testing): 建议 mock selector 或对 use_fp4 用例做 SM100 skip; 最终处理状态未能从 head 版本确认。
- CuTeDSL launcher 统一与未使用代码清理 (design): 以通用 kernel_launcher + launch 方法收尾, 消除重复的包装逻辑。

风险与影响

- 风险:

1. 预热契约回归风险: `VllmTritonJitKernel.launch` 与 `VllmCuTeDSLJitKernel.launch` 的签名都发生了结构性变化, 所有使用 `kernel_launcher` 的既有内核 (不仅是本 PR 迁移的 DSv4 内核) 都依赖新的 `bind_launch_inputs / launch_spec` 语义, 任何遗漏的调用方都会在 `warmup` 或首次运行时暴露问题。
2. `warmup` 键与运行时键不一致: `fused_indexer_q_cutedsl.py` 的 `cos_sin_dtype` 特化在本 PR 中经历了模型 dtype 与 FP32 的争论, 若最终未按实际 RoPE cache dtype 对齐, BF16 模型在 FP32 rope cache 下仍会出现运行时 JIT 编译;
`sparse_attn_compress_cutedsl.py` 的 `store_full_fp8` 对 `fp8_e5m2/fp8_inc` 等变体的覆盖也依赖手工分支, 存在漏配风险。
3. 跨平台破坏风险: P1 修复前的 `warmup` 注册会让 ROCm/XPU 在启动时尝试编译 CUDA-only CuTe 内核, 修复后仍需要 AMD/XPU CI 验证 Triton fallback 分支未被误伤。
4. 测试平台依赖: `tests/models/test_deepseek_v4_nvidia_attention_jit_warmup.py` 依赖 SM100 能力, 非 Blackwell CUDA runner 上 MXFP4 用例可能失败, 需要确认最终的 skip/mock 策略是否合入。
5. 大规模内核迁移回归: `sparse_attn_compress_cutedsl.py` 单文件 +1945/-1606, 涉及大量内核拆分为单例, 最终恢复了被删 wrapper 才保住 `tests/kernels/test_compressor_kv_cache.py`, 说明迁移过程中存在测试与实现不同步的真实风险。 - 影响: 影响范围集中在 DeepSeek V4 的 NVIDIA 推理路径与全局 JIT warmup 基础设施: 对用户而言, DSv4 在 NVIDIA 上启用 `warmup` 后启动期可预编译全部 CuTeDSL 内核, 运行时不再触发 JIT, 且内核行为 (数值、几何、dispatch 启发式) 保持不变; 对系统而言, Triton 与 CuTeDSL 两套 launcher 收敛为单一 `kernel_launcher` 契约, 后续 GLM-5.2/5.3、Gemma 4、gpt-oss、Kimi-K3 等模型的 de-JIT 迁移都可以直接复用; 对团队而言, 这是 Issue #49349 跟踪列表中依赖链较深的一环, 与 #50175、#50176、#53564、#53565、#56323 等多个并行 PR 有 rebase 冲突风险 (PR 历史中多次出现 merge conflict 提示), 需要统一的集成节奏。 - 风险标记: 核心预热契约变更, 跨平台 CUDA gate 风险, `warmup` 键与运行时键不一致风险, 依赖 SM100 能力测试, 大范围内核迁移回归

关联脉络

- PR #49627 Parent draft PR for DSv4 warmup migration: PR body 明确本 PR 的父级 draft PR, 包含整体迁移设计与跟踪信息。
- PR #50175 Dependency: shared JIT warmup infrastructure: PR body 声明 Depends on #50175, 本 PR 构建在其提供的预热框架之上。
- PR #50176 DSv4 warmup migration sibling PR: Issue #49349 跟踪列表中的同系列迁移 PR, 与 #53566 并行推进 DSv4 de-JIT。
- PR #53564 DSv4 warmup migration sibling PR (merged): Issue #49349 跟踪列表中已完成的同系列 PR, 与 #53566 共享同一组 warmup 基础设施与文件。

- PR #53565 DSv4 warmup migration sibling PR (merged): Issue #49349 跟踪列表中已完成的同系列 PR，与 #53566 协作覆盖 DSv4 不同内核路径。
- PR #56323 DSv4 warmup migration follow-up PR: Issue #49349 跟踪列表中的后续迁移项，承接 #53566 未覆盖的内核。
- PR #50109 Enable --jit-monitor-mode error on CI: 本 PR 的最终验收依赖在 CI 上启用 jit-monitor 报错模式，用于捕获未被 warmup 覆盖的运行时内核。