PR #49315 完整报告

vllm-project/vllm

[2/N][Feat][Perf] Add new warmup infrastructure for JITs. Add predicate filtering for JIT warmup, and migrate Inkling FA4

合并时间：2026-08-11 19:46

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/49315

执行摘要

- 一句话：JIT warmup 新增谓词过滤，迁移 Inkling FA4 预热
- 推荐动作：值得精读。重点看三点：_when 谓词的 AST 追踪与静态求值设计、zip_inputs 锁步展开与笛卡尔积的语义区分、以及 InklingFA4RelAttentionKernel.dispatch 如何把运行期参数映射为静态 CompileKey。团队在编写新 kernel warmup 时可照此 PR 作为参考实现；同时关注 legacy CuTeDSL registry 的后续清除（依赖 #50089 合入）。

功能与动机

PR body 明确动机：尊重 enable_jit_warmup 开关并通过 kernel_warmup() 统一执行预热、使用 RFC #47456 的 kernel-owned 共享预热契约、走标准 warmup 日志 / 顺序 / 异常处理路径、保持模型构建廉价且无副作用、避免模型构建期间每层重复编译（即使有缓存）、将运行期执行与启动编译清晰分离。这些目标服务于关联 issue #49349（Zero JIT compilation during runtime），该 issue 将本 PR 标记为已完成的基础工作之一，最终目标是配合 --jit-monitor-mode error 让模型启动阶段完成全部 JIT 预热。

实现拆解

实现分 5 步拆解：

1. 扩展共享 JIT warmup 契约（vllm/model_executor/warmup/jit_warmup.py）：新增 WarmupIntRange.advance 字段支持自定义单调递进（如按 2 的幂增长），避免超大数值网格枚举导致 warmup key 爆炸；把 dispatch 求值逻辑抽成 _eval_local_exprs 供 compile key 与谓词共用；新增 _WarmupPredicateTrace 与 _trace_warmup_predicate，把 _when=<predicate> 参数通过 AST 追踪成可静态求值的谓词；将 _collect_dispatch_body 泛化为 _collect_expression_body，支持任意返回表达式的追踪，并抽出 _function_trace_inputs 统一处理函数签名默认值与绑定 self。
2. 迁移 Inkling FA4 相对注意力（vllm/models/inkling/nvidia/ops/fa4_rel_attention.py、删除 fa4_warmup.py、改 attention.py）：新增 InklingFA4RelAttentionKernel(VllmJitKernel)，提供 CompileKey（is_local、num_heads、num_splits、num_warps_bucket、large_num_reqs 等静态字段）、dispatch(...)（把服务请求级参数映射为编译 key）、_is_valid_warmup_dispatch(...)（谓词过滤）与 get_warmup_keys(vllm_config)（读取调度 / 缓存 /TP 配置生成全部 warmup key）；删除旧 fa4_warmup.py 中的 register_fa4_warmup/_iter_compile_units 注册机制；InklingAttention.__init__ 移除注册

副作用，运行期前向改调 INKLING_FA4_REL_ATTENTION_KERNEL(...)。

3. 简化 FA4 MLA prefill warmup（vllm/v1/attention/backends/mla/prefill/flash_attn.py）：删除 _FA4MLAPrefillShapeProbe dataclass，dispatch(...) 直接接收 max_seqlen_q/max_seqlen_k；shape probe 展开从手写 if 分支改为 zip_inputs(*shape_probes) 锁步展开 + _when=self._is_valid_warmup_shape_probe 谓词过滤，过滤逻辑（long-K、very-long-K、SM90、qk/v head dim 差异等条件）收敛到单一方法；kernel 改为静态方法并支持 runtime_kernel 注入，__call__ 统一走 self.kernel(...)。
4. 统一 warmup 入口（vllm/model_executor/warmup/fa4_cutedsl_warmup.py、cutedsl_warmup.py）：fa4_cutedsl_warmup 拆分为 _warm_fa4_mla_prefill 与 _warm_inkling_fa4_rel_attention 两个子步骤并在入口聚合；legacy cutedsl_warmup.py 仅标记为 deprecated 兼容注册表（docstring 层面），因 Kimi K3 冲突暂不删除运行路径。
5. 测试与配套：tests/model_executor/test_jit_warmup.py 新增 advance 合法性校验与 PredicateKernel 谓词过滤测试；tests/models/inkling/test_fa4_warmup.py 从旧 InklingFA4WarmupConfig 枚举改为基于 SimpleNamespace 构造的 VllmConfig 驱动 get_warmup_keys，并同时枚举 local/global 两层配置做全量一致性断言；test_fa4_rel_attention.py 适配新调用名；vllm/config/kernel.py 与 vllm/models/inkling/nvidia/ops/__init__.py 有配套微调。

关键文件：

- vllm/model_executor/warmup/jit_warmup.py（模块 启动预热；类别 source；类型 data-contract；符号 _eval_local_exprs, _expand_warmup_value_grid, _WarmupPredicateTrace, matches）：共享 JIT warmup 契约的核心扩展点：新增 _when 谓词 AST 追踪（_WarmupPredicateTrace、_trace_warmup_predicate）、WarmupIntRange.advance 自定义递进、_eval_local_exprs 与 _function_trace_inputs 抽取，所有后续 kernel 迁移都依赖此文件的数据契约。
- vllm/models/inkling/nvidia/ops/fa4_rel_attention.py（模块 相对注意力；类别 infra；类型 infrastructure；符号 inkling_fa4_rel_attention, _num_warps_bucket, InklingFA4RelAttentionKernel, CompileKey）：Inkling FA4 相对注意力迁移主战场：新增 InklingFA4RelAttentionKernel 包装（CompileKey/dispatch/get_warmup_keys/_is_valid_warmup_dispatch），运行期调用与启动期编译 key 生成彻底解耦，是迁移样板的核心文件。
- vllm/v1/attention/backends/mla/prefill/flash_attn.py（模块 MLA 预填充；类别 source；类型 core-logic；符号 _FA4MLAPrefillShapeProbe, kernel, _is_valid_warmup_shape_probe）：FA4 MLA prefill warmup 从手写 if 分支重构为 zip_inputs + _when 谓词的声明式展开，并新增可注入 runtime_kernel 的静态 kernel 方法，是谓词能力在真实后端的首个应用点。
- vllm/models/inkling/nvidia/ops/fa4_warmup.py（模块 FA4 预热；类别 infra；类型 deletion；符号 InklingFA4WarmupConfig, _num_warps_bucket, _compile, _iter_compile_units）：删除整个 legacy 注册机制（154 行），register_fa4_warmup/_iter_compile_units/InklingFA4WarmupConfig 不再需要，模型构建期的注册副作用随之移除。

- vllm/model_executor/warmup/fa4_cutedsl_warmup.py（模块 预热入口；类别 source；类型 data-contract；符号 fa4_cutedsl_warmup, _warm_fa4_mla_prefill, _warm_inkling_fa4_rel_attention）：warmup 入口拆分：fa4_cutedsl_warmup 现在聚合 MLA prefill 与 Inkling FA4 相对注意力两个独立预热步骤，是统一预热编排的接线点。
- vllm/models/inkling/nvidia/attention.py（模块 模型注意力；类别 source；类型 data-contract；符号 INKLING_FA4_REL_ATTENTION_KERNEL）：移除 register_fa4_warmup 调用，运行期前向改调 INKLING_FA4_REL_ATTENTION_KERNEL，验证了模型构建副作用消除的落地效果。
- tests/model_executor/test_jit_warmup.py（模块 预热测试；类别 test；类型 test-coverage；符号 test_warmup_range_uses_custom_advancement, test_warmup_range_validates_custom_advancement, test_trace_dispatch_filters_with_traced_predicate, PredicateKernel）：新增 WarmupIntRange.advance 的用法 / 校验测试与 PredicateKernel 谓词过滤测试，直接验证 _when 契约的语义。
- tests/models/inkling/test_fa4_warmup.py（模块 预热测试；类别 test；类型 test-coverage；符号 _vllm_config_from_reference_config, test_warmup_enumerates_every_runtime_compile_class）：测试从旧注册枚举改为基于 VllmConfig 驱动 get_warmup_keys，同时枚举 local/global 两层配置并全量断言 CompileKey 集合，守护预热枚举与运行期编译类的一致性。
- vllm/model_executor/warmup/cutedsl_warmup.py（模块 兼容注册表；类别 source；类型 data-contract；符号 cutedsl_warmup）：被标记为 deprecated 兼容注册表，因 Kimi K3 冲突暂未删除运行路径；review 讨论的 warning 日志未进入最终版，仅保留 docstring 标记。
- tests/models/inkling/test_fa4_rel_attention.py（模块 相对注意测试；类别 test；类型 test-coverage）：正确性测试从 inkling_fa4_rel_attention 切换到 INKLING_FA4_REL_ATTENTION_KERNEL 包装调用，确认运行期计算路径未变。

关键符号：_trace_warmup_predicate, _WarmupPredicateTrace.matches, _eval_local_exprs, _expand_warmup_values, InklingFA4RelAttentionKernel.get_warmup_keys, InklingFA4RelAttentionKernel.dispatch, InklingFA4RelAttentionKernel._is_valid_warmup_dispatch, FA4MLAPrefillKernel.get_warmup_keys, FA4MLAPrefillKernel._is_valid_warmup_shape_probe, FA4MLAPrefillKernel.kernel, _warm_inkling_fa4_rel_attention, fa4_cutedsl_warmup

关键源码片段

vllm/model_executor/warmup/jit_warmup.py

共享 JIT warmup 契约的核心扩展点：新增 _when 谓词 AST 追踪（_WarmupPredicateTrace、_trace_warmup_predicate）、WarmupIntRange.advance 自定义递进、_eval_local_exprs 与 _function_trace_inputs 抽取，所有后续 kernel 迁移都依赖此文件的数据契约。

```
# vllm/model_executor/warmup/jit_warmup.py
```

```
# 共享 JIT warmup 基础设施：本 PR 新增 `_when` 谓词与自定义递进能力。
# 以下三块是核心求值链路，warmup 期间零运行时执行，全部基于 AST 追踪。

# 局部赋值求值器：按顺序求值 trace 到的本地赋值表达式，
# CompileKey 与谓词 matches 共用这条路径，保证两处语义一致。
def _eval_local_exprs(
    local_exprs: _LocalExprs,
    values: Mapping[str, Any],
    globals_: Mapping[str, Any],
) -> dict[str, Any]:
    evaluated_values = dict(values)
    for name, expr in local_exprs:
        evaluated_values[name] = _eval_dispatch_expr(expr, evaluated_values, globals_)
    return evaluated_values


# WarmupIntRange 支持自定义单调递进 advance，例如按 2 的幂增长，
# 避免为超大数值网格逐一枚举导致 warmup key 爆炸。
def _expand_warmup_values(values: WarmupValues) -> tuple[Any, ...]:
    if isinstance(values, WarmupIntRange):
        if values.advance is None:
            # 默认按 range 语义展开：start/stop/step
            return tuple(range(values.start, values.stop, values.step))
        if values.step != 1:
            raise ValueError("WarmupIntRange cannot set both step and advance")
        # 自定义递进要求严格递增，防止死循环或重复 key
        expanded: list[int] = []
        value = values.start
        while value < values.stop:
            expanded.append(value)
            next_value = values.advance(value)
            if next_value <= value:
                raise ValueError("WarmupIntRange.advance must return a greater value")
            value = next_value
        return tuple(expanded)
    if isinstance(values, (list, tuple)):
        return tuple(values)
    return (values,)


# 谓词追踪结果：`_when=<predicate>` 在 warmup 展开前过滤 dispatch 点，
# 只有被生成 compile key 的组合才会进入编译。
@dataclass(frozen=True)
class _WarmupPredicateTrace:
    local_exprs: _LocalExprs
    return_expr: ast.AST
    globals: Mapping[str, Any]
    input_names: frozenset[str]
    defaults: Mapping[str, Any]
```

```
    def matches(self, kwargs: Mapping[str, Any]) -> bool:
        # 与 CompileKey 求值走同一路径：先算本地表达式，再求返回表达式
        dispatch_values = _eval_local_exprs(
            self.local_exprs, {**self.defaults, **kwargs}, self.globals
        )
        return bool(
            _eval_dispatch_expr(self.return_expr, dispatch_values, self.globals)
        )
```

vllm/models/inkling/nvidia/ops/fa4_rel_attention.py

Inkling FA4 相对注意力迁移主战场：新增 InklingFA4RelAttentionKernel 包装（CompileKey/dispatch/get_warmup_keys/_is_valid_warmup_dispatch），运行期调用与启动期编译 key 生成彻底解耦，是迁移样板的核心文件。

```
# vllm/models/inkling/nvidia/ops/fa4_rel_attention.py
# Inkling FA4 相对注意力迁移到 VllmJitKernel 契约：
# dispatch() 把「服务请求级」参数映射为「编译静态」CompileKey，
# 该函数体由 _trace_dispatch 做 AST 追踪，warmup 期不真正执行。

def dispatch(
    self,
    *,
    is_local: bool,
    num_heads: int,
    num_kv_heads: int,
    head_dim: int,
    rel_extent: int,
    dtype: torch.dtype,
    kv_dtype: torch.dtype,
    block_size: int,
    window_size: tuple[int, int],
    max_kv_len: int,
    query_len: int,
    num_reqs: int,
) -> CompileKey:
    # 运行时的分桶与 split 决策在 warmup 期被静态复算，
    # 保证预热 key 与运行期实际 key 完全一致
    max_seqlen_q = bucket_max_seqlen_q(query_len)
    num_splits = inkling_fa4_num_splits(
        is_local=is_local,
        batch_size=num_reqs,
        max_query_len=max_seqlen_q,
        num_heads=num_heads,
        num_kv_heads=num_kv_heads,
        max_kv_len=max_kv_len,
    )
    return self.CompileKey(
        is_local=is_local,
        num_heads=num_heads,
```

```
        num_kv_heads=num_kv_heads,
        head_dim=head_dim,
        rel_extent=rel_extent,
        dtype=dtype,
        kv_dtype=kv_dtype,
        block_size=block_size,
        window_size=window_size,
        max_seqlen_q=max_seqlen_q,
        num_splits=num_splits,
        num_warps_bucket=(_num_warps_bucket(num_reqs) if num_splits > 1 else None),
        large_num_reqs=num_reqs > 1024,
    )


# `_when` 谓词：query_len 与 num_reqs 组合超出调度上限时直接丢弃，
# 避免为运行期不可能出现的形状编译 kernel。
def _is_valid_warmup_dispatch(
    self,
    *,
    query_len: int,
    num_reqs: int,
    max_num_batched_tokens: int,
) -> bool:
    return query_len + num_reqs <= max_num_batched_tokens + 1
```

vllm/v1/attention/backends/mla/prefill/flash_attn.py

FA4 MLA prefill warmup 从手写 if 分支重构为 zip_inputs + _when 谓词的声明式展开，并新增可注入 runtime_kernel 的静态 kernel 方法，是谓词能力在真实后端的首个应用点。

```
# vllm/v1/attention/backends/mla/prefill/flash_attn.py
# FA4 MLA prefill warmup 由「手写 if 分支扩展 shape probe」改为
# zip_inputs + _when 谓词，过滤逻辑收敛到一个方法里。

# 谓词：仅在 FA4 实际会命中对应 kernel 路径的 shape 上保留编译 key，
# 例如 SM90 的 very-long-K 路径要求 qk_head_dim 与 v_head_dim 不同。
def _is_valid_warmup_shape_probe(
    self,
    *,
    max_seqlen_k: int,
    num_splits: int,
    is_sm90: bool,
    qk_head_dim: int,
    effective_v_head_dim: int,
) -> bool:
    long_k = FA4_MLA_PREFILL_LONG_K_BLOCKS * FA4_MLA_PREFILL_K_TILE
    very_long_k = FA4_MLA_PREFILL_VERY_LONG_K_BLOCKS * FA4_MLA_PREFILL_K_TILE
    return (
        max_seqlen_k < long_k
        or (num_splits != 1 and max_seqlen_k == long_k)
```

```
        or (
            num_splits != 1
            and max_seqlen_k == very_long_k
            and not is_sm90
            and qk_head_dim != effective_v_head_dim
        )
    )
```

```
# get_warmup_keys 尾部：zip_inputs 把多行 probe 按行锁步展开（不是笛卡尔积），
# _when 在展开前过滤无效 dispatch 点，避免编译不可能出现的组合。
return self._trace_dispatch(self.dispatch)(
    zip_inputs(*shape_probes),
    batch_size=FA4_MLA_PREFILL_COMPILE_BATCH_SIZE,
    dtype=dtype,
    num_heads=num_heads,
    mla_dims=mla_dims,
    requires_v_padding=requires_v_padding,
    is_sm90=is_sm90,
    qk_head_dim=qk_head_dim,
    effective_v_head_dim=effective_v_head_dim,
    causal=FA4_MLA_PREFILL_CAUSAL_OPTIONS,
    return_lse=FA4_MLA_PREFILL_LSE_OPTIONS,
    num_splits=num_splits,
    fa_version=fa_version,
    _when=self._is_valid_warmup_shape_probe,
)
```

评论区精华

评审主要由 LucasWilkinson 主导，共 3 条行内评论，均为非阻塞问题，最终获得 LucasWilkinson 与 MatthewBonanni 双 APPROVED：

- cutedsl_warmup.py 新增 deprecation warning 日志时，LucasWilkinson 质疑："will this log trigger for regular users? regular users should be blind to this refactor"。最终合并版未包含运行期 warning，仅保留 docstring 弃用标记（该文件最终仅 +1/-1），符合 reviewer 对普通用户无感的要求。
- jit_warmup.py 中 _expand_warmup_value_axes 与 _function_trace_inputs 均被问 "is this only used in one place? if its just a 1-liner I think we can inline it"。属于 nit 级简化建议，未阻塞合并；_function_trace_inputs 仍保留在最终代码中，说明作者选择了保留独立函数以维持可读性。
- deprecation warning 是否会打扰普通用户 (design): 最终合并版未包含运行期 warning，该文件仅保留 docstring 层面的弃用标记（+1/-1），符合 reviewer 对普通用户无感的要求。
- _expand_warmup_value_axes 是否应内联 (style): nit 级简化建议，未阻塞合并；PR 已由两位 reviewer approve。
- _function_trace_inputs 是否应内联 (style): nit 级建议；最终代码仍保留 _function_trace_inputs 独立函数，作者选择保留可读性。

风险与影响

- 风险：
  1. AST 追踪解析边界（jit_warmup.py）：_when 谓词与 dispatch 一样只支持受限 Python 子集（本地赋值、算术、比较、布尔表达式、无 **kwargs 的函数调用），一旦未来谓词引入外部可变状态或复杂控制流，matches() 可能与真实运行语义脱节，且以 ValueError 在启动期直接失败。
  2. 过滤逻辑与 FA4 运行时 selector 的耦合（flash_attn.py、fa4_rel_attention.py）：_is_valid_warmup_shape_probe 的 very-long-K 分支依赖 not is_sm90 and qk_head_dim != effective_v_head_dim 等条件，与 FA4 内部选择器强耦合；若上游调整分桶策略而此处未同步，会产生缺失的 warmup key，导致运行期重新编译——正是 #49349 要消除的反模式。
  3. 双 warmup 路径并存窗口：因 #50089（Kimi K3）冲突，legacy CuTeDSL 注册表与新的 kernel.warmup() 路径并存，未迁移的 provider 存在重复预热或遗漏覆盖的过渡期风险。
  4. 模型构建副作用移除（attention.py）：InklingAttention.__init__ 不再注册 warmup，全部延后到 worker 阶段依赖完整 vllm_config，若未来模型在配置未定型时触发 attention 前向会失去预热保护。
  5. 测试配置覆盖单一：test_warmup_enumerates_every_runtime_compile_class 只覆盖 16 heads/2 kv heads 的固定配置，异构 TP 分片、不同 kv head 分配下的枚举一致性依赖 CI 补充。 - 影响：对用户：启用 enable_jit_warmup 的 Inkling 模型与服务启动路径的 warmup 顺序、日志、异常处理被标准化，模型构建阶段不再逐层编译；普通用户不感知重构（reviewer 特意把关）。对系统：FA4 MLA prefill 与 Inkling FA4 两个后端的预热 key 数量因谓词过滤而收敛，启动编译量下降。对团队：jit_warmup.py 成为 kernel 迁移样板（VllmJitKernel + zip_inputs + _when + WarmupIntRange.advance），#49349 中 DeepSeek V4、Triton、TileLang 等后续迁移可直接复用同一契约。影响程度中高：运行期计算路径不变，但两项 attention 后端的启动期行为被改写。 - 风险标记：AST 追踪解析边界，核心启动路径变更，双 registry 并存窗口，过滤与运行时 selector 耦合，测试配置覆盖单一

关联脉络

- PR #47451 Shared JIT warmup contract（#47451，PR body 引用）：PR body 明确本 PR 是基于 #47451 引入的共享 JIT warmup 契约落地的第 2 步（2/N），本 PR 为其扩展 _when 谓词与 advance 能力。
- PR #50089 Kimi K3 integration（与 legacy CuTeDSL deprecation 冲突）：PR body 说明 legacy CuTeDSL warmup 路径的弃用因与 Kimi K3 集成冲突而无法在本 PR 完成，将延后到未来 PR 处理。
- PR #46167 Add --jit-monitor-mode（issue #49349 引用）：issue #49349 将 #46167 与本 PR 列为 de-JITification 基础工作；该模式可在 CI 检测运行期未追踪编译，是本 PR 目标的验证手段。