PR #50174 完整报告

vllm-project/vllm

[3/N][Feat][Perf] Add new warmup infrastructure for JITs. Add provider registry and orchestration for JIT warmup

合并时间：2026-08-18 01:11

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/50174

执行摘要

- 一句话：新增 JIT warmup 注册表与编排，迁移 block-table 内核预热
- 推荐动作：值得精读。这是 vLLM 启动期 JIT 预热架构的地基 PR，重点看三处设计：VllmJitKernel 的 CompileKey/dispatch/get_warmup_keys 契约如何让预热 key 与运行期特化严格一致；named_parameters 对 **kwargs 转发的支持如何在静态可见性与低样板之间取舍；triton_scalar_specialization_rep 对 Triton 缓存键类的精确建模。合并前建议复核 kernel_warmup() 中非 v2 runner 分支与新增 registry 编排的执行顺序，以及 enable_jit_warmup 默认值对老路径的兼容性。

功能与动机

PR body 明确动机：从实际实例化的 model 与 backend 对象中发现 warmup provider，避免基于模型名清单预热未激活后端的 kernel；将编译纳入 kernel_warmup() 的日志、排序、进度与异常处理框架；尊重 enable_jit_warmup；保持模型构建与运行时执行和启动期编译分离。关联 issue #49349 提出 "Zero JIT compilation during runtime" 总目标，本 PR 被列为基础阻塞项之一。

实现拆解

1. 建立注册契约与运行时缓存：在 vllm/model_executor/warmup/jit_warmup.py 中新增 JitWarmupRegistry（基于 ContextVar 的作用域激活，register_warmup() 只记录元数据不编译，warmup() 统一展开、去重并调用 VllmJitKernel.warmup()）；VllmJitKernel 新增 _get_or_compile() 运行时缓存，compile() 未写缓存时抛 RuntimeError，把 " 运行时隐性编译 " 变成可监控的显式错误。
2. 扩展 AST 追踪能力：_DispatchExprEvaluator 新增 visit_Subscript、Python builtins 解析、in/not in/is/is not 比较；get_function_source_node 支持 lambda；_trace_compile_key_dispatch 支持 dispatch 通过自身 **kwargs 参数一次性转发 CompileKey 字段（named_parameters 机制），重复或未知字段在构造 CompileKey 时抛 TypeError。
3. 迁移 block-table 内核：vllm/v1/worker/block_table.py 中 _compute_slot_mapping_kernel 被封装成 ComputeSlotMappingKernel(VllmJitKernel)，BlockTable.__init__ 在 SlotMappingMode.TOKEN_TO_KV_SLOT 下调用 register_warmup() 快照实际特化标量；compute_slot_mapping() 改为走统一 kernel 调用

；同步删除 vllm/model_executor/warmup/v1_block_table_warmup.py 的旧预热函数。

4. 接入启动编排：GPUModelRunner 与 v1 GPUModelRunner(gpu/model_runner.py) 持有 jit_warmup_registry，两处 InputBatch 构建（占位与真实 KV-cache 几何确定后）都包在 activate() 作用域内；kernel_warmup() 在 enable_jit_warmup 开启时调用 registry.warmup()，带 perf_counter 计时与异常日志，且异常会 re-raise。
5. 配套：jit_warmup_triton_helper.py 新增 triton_scalar_specialization_rep；新增 docs/contributing/jit_kernel_warmup.md（约 400 行契约与搜索空间参考）；测试覆盖注册表去重、**kwargs 转发、内置函数解析、下标表达式、标量特化代表值及迁移前后 dispatch 一致性。

关键文件：

- vllm/model_executor/warmup/jit_warmup.py（模块 预热框架；类别 source；类型 data-contract；符号 JitWarmupRegistry, register_warmup, activate, input_names_for）：核心契约文件：新增 JitWarmupRegistry、扩展 AST 追踪（builtins/Subscript/in 比较 / 命名参数转发）、VllmJitKernel 增加 _get_or_compile 运行时缓存，是所有预热接入方必须遵守的地基。
- vllm/v1/worker/block_table.py（模块 块表预热；类别 source；类型 core-logic；符号 ComputeSlotMappingKernel, CompileKey, dispatch, get_warmup_keys）：迁移主体：裸 Triton 内核 _compute_slot_mapping_kernel 封装为 ComputeSlotMappingKernel，BlockTable 构造时注册实际特化参数，是本次第一个真正接入新契约的运行时内核。
- vllm/model_executor/warmup/kernel_warmup.py（模块 启动编排；类别 source；类型 core-logic；符号 kernel_warmup）：统一编排入口：把 registry.warmup() 接入既有 kernel_warmup()，带计时、日志与异常 re-raise，并受 enable_jit_warmup 门控。
- vllm/v1/worker/gpu_model_runner.py（模块 模型加载；类别 source；类型 entrypoint；符号 JitWarmupRegistry, activate）：接线点：GPUModelRunner 持有 JitWarmupRegistry，并在两处 InputBatch 构建（占位与真实 KV-cache 几何）外包裹 activate() 作用域收集注册。
- vllm/model_executor/warmup/jit_warmup_triton_helper.py（模块 特化辅助；类别 source；类型 data-contract；符号 triton_scalar_specialization_rep）：新增 triton_scalar_specialization_rep，把 int 映射为 Triton 缓存键三类代表值，是保证预热 key 与运行期特化一致的关键帮助函数。
- vllm/model_executor/warmup/v1_block_table_warmup.py（模块 旧预热路径；类别 source；类型 deletion；符号 warm_v1_block_table_kernels）：被删除的旧手工预热路径，功能完全被 ComputeSlotMappingKernel.compile() 取代，删减 29 行。
- docs/contributing/jit_kernel_warmup.md（模块 贡献文档；类别 docs；类型 documentation；符号 MyKernel, _is_valid_warmup_input）：新增贡献者契约文档，定义 kernel wrapper、CompileKey、dispatch、get_warmup_keys、compile 的职责边界与搜索空间语法，是后续迁移的指南。
- tests/model_executor/test_jit_warmup.py（模块 预热测试；类别 test；类型 test-coverage；符号 test_triton_scalar_specialization_rep, test_dispatch_helper_calls_resolve_python_builtins, test_dispatch_can_forward_compile_key_fields,

test_registry_records_only_inside_model_setup_context）：核心测试：覆盖 triton_scalar_specialization_rep 边界、builtins 解析、下标表达式、**kwargs 转发、注册表作用域与去重、运行时缓存 miss 编译。

- tests/v1/worker/test_jit_warmup_migration.py（模块 迁移验证；类别 test；类型 test-coverage；符号 test_compute_slot_mapping_warmup_matches_runtime_specializations）：迁移一致性验证：断言 ComputeSlotMappingKernel 的 dispatch 与 get_warmup_keys 在不同 block_table 几何下产生与 Triton 特化一致的 CompileKey。
- vllm/v1/worker/gpu/model_runner.py（模块 模型加载；类别 source；类型 entrypoint；符号 JitWarmupRegistry）：v2 模型侧的平行接线：ModelRunner 同样新增 jit_warmup_registry，保证后续模型级预热接入点存在。
- vllm/v1/worker/cpu_model_runner.py（模块 模型加载；类别 source；类型 entrypoint；符号 JitWarmupRegistry）：CPU runner 仅添加 registry 实例，尚未接入 warmup 编排，可能是后续补齐点。
- docs/contributing/README.md（模块 贡献文档；类别 docs；类型 documentation）：贡献文档索引补充，让新文档可被导航发现。

关键符号：JitWarmupRegistry.activate, JitWarmupRegistry.register_warmup, JitWarmupRegistry.warmup, VllmJitKernel._get_or_compile, VllmJitKernel.compile_key, _CompileKeyDispatchTrace.input_names_for, _CompileKeyDispatchTrace.compile_key, _DispatchExprEvaluator.visit_Subscript, get_function_source_node, triton_scalar_specialization_rep, ComputeSlotMappingKernel.dispatch, ComputeSlotMappingKernel.get_warmup_keys, ComputeSlotMappingKernel.compile, kernel_warmup

关键源码片段

vllm/model_executor/warmup/jit_warmup.py

核心契约文件：新增 JitWarmupRegistry、扩展 AST 追踪（builtins/Subscript/in 比较 / 命名参数转发）、VllmJitKernel 增加 _get_or_compile 运行时缓存，是所有预热接入方必须遵守的地基。

```
# vllm/model_executor/warmup/jit_warmup.py
# _CompileKeyDispatchTrace.compile_key：在静态追踪 dispatch 时，
# 允许把 dispatch 自身 **kwargs 里未被声明的命名参数 1:1 转发给 CompileKey 字段。
def compile_key(
    self,
    compile_key_type: type[CompileKeyT],
    kwargs: Mapping[str, Any],
) -> CompileKeyT:
    dispatch_values = _eval_local_exprs(
        self.local_exprs, {**self.defaults, **kwargs}, self.globals
    )
    named_parameters = self.named_parameters
    # 先物化直接转发的字段（不在命名参数集合里的 kwargs 直通 CompileKey）。
    fields: dict[str, Any] = {}
    if named_parameters is not None:
```

```
        fields = {
            name: value
            for name, value in kwargs.items()
            if name not in named_parameters
        }
    # 命名参数则按 AST 表达式求值（可参与 range 展开、特化映射等变换）。
    for field, expr in self.field_exprs:
        if field in fields:
            raise TypeError(f"CompileKey field '{field}' is specified twice")
        fields[field] = _eval_dispatch_expr(expr, dispatch_values, self.globals)
    return compile_key_type(**fields)


# VllmJitKernel 运行时入口：cache miss 时先编译，再检查是否真的写入了缓存，
# 否则抛 RuntimeError，把 " 运行时隐性 JIT 编译 " 变成可被 jit-monitor 捕获的显式错误。
def _get_or_compile(
    self,
    compile_key: CompileKeyT,
    *,
    runtime_context: Mapping[str, Any] | None = None,
) -> Any:
    if compile_key not in self._compiled_cache:
        self.compile(compile_key)
    try:
        return self._compiled_cache[compile_key]
    except KeyError as exc:
        details = [f"compile_key={compile_key!r}"]
        if runtime_context:
            details.append(f"runtime_context={dict(runtime_context)!r}")
        raise RuntimeError(
            f"{type(self).__name__}.compile(...) did not cache its JIT "
            f"executor ({', '.join(details)})"
        ) from exc
```

vllm/v1/worker/block_table.py

迁移主体：裸 Triton 内核 _compute_slot_mapping_kernel 封装为 ComputeSlotMappingKernel，BlockTable 构造时注册实际特化参数，是本次第一个真正接入新契约的运行时内核。

```
# vllm/v1/worker/block_table.py
# 迁移后的 slot-mapping 内核：从裸 Triton 函数变成 VllmJitKernel 子类，
# 统一暴露 CompileKey / dispatch / compile 契约，供注册表在启动期预热。
class ComputeSlotMappingKernel(VllmJitKernel["ComputeSlotMappingKernel.CompileKey"]):
    triton_block_size = 1024

    @dataclass(frozen=True)
    class CompileKey:
        # 只有真正参与 Triton 特化的标量才进 key；
        # num_tokens / max_num_tokens 已在 do_not_specialize 中排除。
```

```
    kv_cache_block_size: int
    blocks_per_kv_block: int
    total_cp_world_size: int
    total_cp_rank: int
    cp_kv_cache_interleave_size: int
    block_table_stride: int
    block_size: int

def dispatch( # type: ignore[override]
    self,
    *,
    block_table_stride: int,
    block_size: int,
    **compile_key_fields: int,
) -> CompileKey:
    # 直接字段走 **compile_key_fields 转发；stride / block_size 则映射成
    # Triton 特化类的代表值（1、16 的倍数或 generic 代表）。
    return self.CompileKey(
        **compile_key_fields,
        block_table_stride=triton_scalar_specialization_rep(block_table_stride),
        block_size=triton_scalar_specialization_rep(block_size),
    )

def get_warmup_keys(self, **dispatch_kwargs: int) -> list[CompileKey]:
    # 复用 dispatch 的 AST 追踪，保证预热 key 与运行期 dispatch 完全一致。
    return self._trace_dispatch(self.dispatch)(**dispatch_kwargs)

def compile(self, compile_key: CompileKey) -> None:
    # compile-only 预热：用伪张量描述符触发 Triton 编译，不分配真实 buffer。
    warmup = getattr(self.kernel, "warmup", None)
    assert warmup is not None
    int32_ptr = TritonWarmupTensor(torch.int32)
    int64_ptr = TritonWarmupTensor(torch.int64)
    warmup(
        2, # 任意值即可，num_tokens / max_num_tokens 在 do_not_specialize 中
        2,
        int32_ptr, int64_ptr, int32_ptr,
        compile_key.block_table_stride,
        compile_key.block_size,
        int64_ptr,
        KV_CACHE_BLOCK_SIZE=compile_key.kv_cache_block_size,
        BLOCKS_PER_KV_BLOCK=compile_key.blocks_per_kv_block,
        TOTAL_CP_WORLD_SIZE=compile_key.total_cp_world_size,
        TOTAL_CP_RANK=compile_key.total_cp_rank,
        CP_KV_CACHE_INTERLEAVE_SIZE=compile_key.cp_kv_cache_interleave_size,
        PAD_ID=PAD_SLOT_ID,
        BLOCK_SIZE=self.triton_block_size,
        grid=(2,),
    )
```

vllm/v1/worker/gpu_model_runner.py

接线点：GPUModelRunner 持有 JitWarmupRegistry，并在两处 InputBatch 构建（占位与真实 KV-cache 几何）外包裹 activate() 作用域收集注册。

```
# vllm/v1/worker/gpu_model_runner.py
# 两处 InputBatch 构建都包在 registry 作用域内：模型 / KV-cache 组件
# 在激活期间调用 register_warmup() 只记录元数据，不触发编译。
self.jit_warmup_registry = JitWarmupRegistry(vllm_config)

# 占位 InputBatch（模型加载前、KV-cache 几何未知时）
# Capture warmup providers registered by the initial placeholder InputBatch
with self.jit_warmup_registry.activate():
    self.input_batch = InputBatch(
        max_num_reqs=self.max_num_reqs,
        max_model_len=max(self.max_model_len, self.max_encoder_len),
        max_num_batched_tokens=self.max_num_tokens,
        device=self.device,
        vocab_size=self.model_config.get_vocab_size(),
        block_sizes=[placeholder_block_size],
        kernel_block_sizes=[placeholder_block_size],
        max_num_blocks_per_req=[placeholder_max_num_blocks],
        # … 其余参数与迁移前保持一致
    )

# may_reinitialize_input_batch() 中真实 KV-cache 几何确定后再次捕获注册。
with self.jit_warmup_registry.activate():
    self.input_batch = InputBatch(
        # … 携带最终 block_sizes / slot_mapping_modes 等真实参数
    )
```

评论区精华

核心讨论围绕 " 如何降低采用成本 " 与 "AST 静态可见性的取舍 " 展开，结论均已落入代码：

- LucasWilkinson 问 InputBatch.__init__ 期间会调用哪些 JIT kernel，LopezCastroRoberto 澄清此时只有 ComputeSlotMappingKernel 被注册，activate() 是为后续 50175-50178 预留；Lucas 一度建议暂时跳过，最终保留并写入注释。
- 样板代码争议：Lucas 担心 dispatch() 大量显式字段降低采用意愿，提出能否用 **kwargs；Roberto 解释 _trace_dispatch() 是静态解析 dispatch 并展开符号化输入，Python 不会替我们做 kwargs 绑定，因此新增 named_parameters 支持，使 **compile_key_fields 变成 1:1 的 CompileKey 字段转发，既精简又保留类型校验。
- get_warmup_keys 统一接收 vllm_config 的提议被否决：该内核依赖最终化 BlockTable 状态，传 self 会让 registry 保留过期 placeholder buffers，快照标量更安全。
- triton_scalar_specialization_rep 的魔法值（1、16、2^31±1 等）依据 Triton 的 handle_long_type() 实现，对应 i32/i64/u64 与 constant-1/divisible-16/generic 三类缓存键。

- 文档术语 "owner" 改为 "kernel wrapper"，并澄清 statement 级 if 与条件表达式支持边界及 unused_input 示例语义。
- InputBatch.__init__期间有哪些 JIT kernel 被调用 (question): Lucas 曾建议先跳过，最终保留；代码注释明确标注两处 activate() 的作用与归属。
- dispatch 样板代码可否用 **kwargs 消减 (design): 通过 named_parameters 实现直接转发，既保持静态可见性又减少样板；附 ForwardingKernel 测试验证。
- get_warmup_keys 是否统一接收 vllm_config (design): 维持注册时快照标量的方案；多数内核后续只需 KERNEL.register_warmup() 无参调用。
- triton_scalar_specialization_rep 的魔法值来源 (documentation): docstring 补充 Triton 缓存键语义说明，测试覆盖全部边界。
- 文档中 "owner" 术语语义不清 (documentation): 术语改为 kernel wrapper，文档已更新。
- traced dispatch 中 statement 级 if 的限制 (documentation): 文档澄清 AST 追踪边界，并说明可借助 helper 函数绕过。
- 文档 AI 换行与 debug_probe 示例名 (style): 文档清理完成，整体篇幅也按 Lucas 建议精简。

风险与影响

- 风险：
  1. 启动路径行为变化：kernel_warmup() 在 enable_jit_warmup 开启时执行 registry.warmup()，任一带注册内核 compile() 抛错会直接中断启动（fail-fast 是设计意图），但可能把过去 " 预热失败仍可运行 " 的场景变成启动失败，需在 CI 覆盖无对应后端库的环境。
  2. 块表内核迁移风险：compute_slot_mapping() 的调用方式从裸 Triton launch（显式传 PAD_ID、BLOCK_SIZE 等 constexpr）改为 wrapper 调用，常量固化在 ComputeSlotMappingKernel 内部；任何依赖旧 kwargs 形态的外部调用方会被破坏。block_table_stride 依赖实际分配，占位注册与真实注册会产生不同 key，靠 registry 去重保障。
  3. AST 追踪契约收紧：_trace_dispatch 拒绝语句级 if、非自身 **kwargs 展开；存量 dispatch 写法若不合规会在启动期抛 ValueError，需在迁移 PR 中逐一适配。
  4. _get_or_compile 缓存契约：compile() 若不写入 _compiled_cache，运行时 cache miss 直接抛 RuntimeError，所有后续 VllmJitKernel 子类都必须遵守。
  5. 平台覆盖：test_jit_warmup_migration.py 明确 skip 非 CUDA 环境，CPU/ROCm 的标量特化边界未在本 PR 验证；cpu_model_runner.py 只挂 registry 未接 warmup，存在遗漏风险。 - 影响：对运行时 / 用户：启动阶段新增统一的 "JIT kernel warmup starting/finished" 日志与计时，为消除首请求 JIT 编译卡顿铺路；除 block-table 内核外，现有模型运行路径对用户基本透明。对系统：所有 v1 worker（GPU/CPU runner）构造时新增 registry 实例，kernel_warmup() 成为唯一启动编译编排入口，旧的手工预热路径被删除。对团队：确立 VllmJitKernel 贡献者契约，后续 DSv4 de-JITification（#50175-50178）与 Triton/TileLang/CuTeDSL 覆盖工作都依赖此基座，是 "Zero JIT compilation during runtime" 里程碑的分水岭。 - 风险标记：核心启动路径变更，新建

数据契约，涉及多 worker 路径，后续 PR 依赖此基础，AST 追踪契约约束，启动失败语义收紧

关联脉络

- PR #49315 [Warmup] Shared JIT warmup infrastructure (foundation): PR body 明确 "builds on #49315"，本 PR 在其基础上扩展注册表与编排，是同一 warmup 基础设施系列的前序。
- PR #49627 [Warmup] DSv4 de-JITification (parent draft): PR body 指明父级 draft PR，本 PR 是其中基础设施层，配合完成 DeepSeek V4 的运行时去 JIT。
- PR #50175 DSv4 de-JITification follow-up (tracked in #49349): review 中明确 activate() 与 registry 能力将被 50175/50176/50177/50178 四个分解 PR 使用；此处标题为按 issue 推断的占位描述。
- PR #52740 [Warmup] Additional DSV4 kernel warmup coverage: chaunceyjiang 在评论中提交的补充预热 PR，经讨论确认功能将被 50178 覆盖后关闭，说明社区已开始基于本基础设施补覆盖。
- PR #46167 [JIT] Add --jit-monitor-mode: issue #49349 中要求模型完全去 JIT 后以 --jit-monitor-mode error 在 CI 兜底，本 PR 的 _get_or_compile 与注册表正是配合该监控模式的运行时防线。
- PR #47456 [Warmup] JIT warmup design contract: PR body 说明本 PR 实现的是 #47456 描述的 warmup 契约，属于同一设计文档的落地。