

PR #51700 完整报告

vllm-project/vllm

[2/2][Model Runner V2] FULL CUDA graph capture for microbatched steps (DBO)

合并时间: 2026-09-14 13:42

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/51700>

执行摘要

- 一句话: MRV2 DBO 微批步骤接入 FULL CUDA 图捕获与回放
- 推荐动作: 值得精读, 尤其是三类设计决策: (a) `_maybe_ubatch_twin` 用与 DP 握手完全相同的阈值函数生成候选, 把“可捕获”与“可分发”强行绑定, 这是避免多 rank 不对称执行的关键; (b) `run()` 拆成 `begin_capturable_run()` / `finish()`, 把线程创建、device 设置与 cuBLAS 预热留在图外, 只让 handoff 与 join 进图, 这是 DBO 能捕获成功的前提; (c) `_capture_stream` 强制捕获流与微批线程 compute stream 一致, 是移植 V1 `gpu_ubatch_wrapper.py::_capture_ubatches` 时最容易漏掉的一点。阅读时请结合 #50945 一起看 (本 PR 的 diff 建立在它之上), 并重点关注 `dp_utils.py` 中 `2 * min(num_tokens_across_dp) < ubatch_desc.num_tokens` 这一守卫——它已被确认存在边界 bug, 跟进补丁前不建议在生产 DP 负载高度不均的场景依赖 FULL 微批回放。此外建议跟踪 coderabbitai 提出的“永不被回放的孪生图”是否会形成显存浪费的后续处理。

功能与动机

动机来自 RFC #50738: DBO 是 DP+EP 部署中隐藏专家 all-to-all 通信延迟的关键手段, 但它此前只存在于 V1 model runner (`gpu_ubatch_wrapper.py` / `ubatching.py`), Model Runner V2 完全没有 DBO 路径, RFC 中明确指出 `_get_v2_model_runner_unsupported_features()` 会在 `--enable-dbo` 与 V2 组合时直接报错, 是 DP+EP 部署迁移到 V2 的真实阻塞。第一段 #50945 让微批步骤在 V2 上以 eager 方式跑通, 但 PR body 指出瓶颈仍在 CPU: "at replay there are no microbatch threads, no threading.Event handoffs and no per-microbatch Python, so the CPU launch path that dominates eager decode disappears". 本 PR 因此把捕获与回放补齐, 同时保持 PIECEWISE + DBO 不支持 (与 V1 runner 一致)。

实现拆解

1. 捕获候选生成 (`vllm/v1/worker/gpu/cudagraph_utils.py`)。
`CudaGraphManager.__init__` 新增 `ubatch_runner` 参数; 新增 `_maybe_ubatch_twin(desc)`, 对每个 FULL 候选派生一个 `num_ubatches=N` 的孪生描述符, 条件是: 存在 `ubatch_runner`、`desc.cg_mode == FULL`、`num_reqs` 不为 `None`、`num_tokens % num_reqs == 0` 且与 `desc.uniform_token_count` 自洽、`num_tokens >= num_ubatches`、并通过 `check_ubatch_thresholds(parallel_config, num_tokens, uniform_decode=True)`。它在 `uniform-decode` 分支与 `mixed` 分支各调用一次; PR body

特别说明 mixed 分支是必需的，因为 MLA 这类只支持 uniform batch 的后端会被降级为 FULL_AND_PIECEWISE，其 FULL 图本来就是 uniform-decode 图，只在 mixed 分支生成会导致这些模型一张微批图都捕获不到。关键是这套判据必须是配置的纯函数：各 rank 在 all-reduce 之后独立分发，若某个 rank 回放图而另一个走 eager，专家 all-to-all 会挂死。

2. 捕获流与线程生命周期 (ubatch_utils.py + cudagraph_utils.py)。UBatchRunner 新增 capture_stream, run() 被拆成 begin_capturable_run() (创建微批线程、设置 device index、预热 cuBLAS workspace、等待 ready barrier, 全部在图外) 与返回的 finish() 回调 (cpu_wait_event.set() 加 join, 在图内), 并新增 abort_pending_run() 供捕获失败时释放被 park 的线程; CudaGraphManager._capture_stream(desc) 让 torch.cuda.graph(..., stream=...) 使用微批线程的 capture_stream, capture() 用 ExitStack 注册 abort_pending_run 兜底。prepare() 新增 cg_mode 与 for_capture 参数, FULL 下按 padded size 构建每微批 attention metadata, 保证捕获与回放的形状和缓冲地址一致。ModelCudaGraphManager.create_forward_fn 新增 ubatch_forward_fn 分支, 并从 store_capture_output 把输出拷进持久缓冲。
3. 分发与 DP 同步 (dp_utils.py + model_runner.py)。
sync_cudagraph_and_dp_padding 的微批分支不再硬编码 cg_mode=NONE, 改为在 cudagraph_manager is not None 时调用 dispatch(num_reqs, ubatch_num_tokens, synced_uniform_token_count, num_active_loras=..., num_ubatches=...), 并新增空第二微批守卫 $2 * \text{int}(\text{num_tokens_across_dp}.\text{min}()) < \text{ubatch_desc}.\text{num_tokens}$; 随后按图描述符刷新 num_tokens, FULL 时用 ubatch_desc.num_reqs 覆盖 num_reqs (带 assert), DPSyncState.eager 由 ubatch_desc.cg_mode == NONE 决定。
GPUModelRunner 把 self.ubatch_runner 传给 ModelCudaGraphManager, 并在 ubatch_runner.prepare(...) 调用处传入 cg_mode=batch_desc.cg_mode、for_capture=dummy_run and batch_desc.cg_mode == FULL。
4. 配置校验放开 (vllm/config/vllm.py)。
_get_dbo_unsupported_features() 由“任何非 NONE 的 cudagraph_mode 都拒绝”改为只拒绝 CUDAGraphMode.PIECEWISE, 注释写明“DBO captures FULL graphs only”; 因此此前被静默拒绝的 --enable-dbo + FULL 组合现在能真正生效。
5. 测试配套 (tests/v1/worker/test_gpu_ubatch_slicing.py)。新增
test_capturable_run_replays_as_a_cudagraph (真实用 torch.cuda.graph(graph, stream=runner.capture_stream) 捕获后 graph.replay(), 验证回放读取被刷新的持久输入缓冲)、test_captured_split_survives_a_replay_with_enough_requests (回放必须保留捕获时的 request 切分)、test_microbatched_graphs_are_only_offered_to_uniform_batches、test_microbatched_graph_needs_every_rank_to_reach_the_split, 并给 _sync_dp 增加 cudagraph_manager 注入点。

关键文件:

- vllm/v1/worker/gpu/cudagraph_utils.py (模块 图捕获管理; 类别 source; 类型 core-logic; 符号 _maybe_ubatch_twin, _capture_stream, store_capture_output, ubatch_forward_fn): 本 PR 的核心: 新增微批候选生成 _maybe_ubatch_twin、捕获流选择 _capture_stream、ubatch_forward_fn 捕获分支与 store_capture_output 持久输出拷贝, 并让 CudaGraphManager / ModelCudaGraphManager 接收 ubatch_runner。

- `vllm/v1/worker/gpu/ubatch_utils.py` (模块 微批执行; 类别 source; 类型 dependency-wiring; 符号 `UBatchRunner.begin_capturable_run`, `UBatchRunner.run`, `abort_pending_run`, `UBatchRunner.prepare`): 把原来的 `run()` (线程创建 + barrier + join 一体化) 拆成图外的 `begin_capturable_run()` 与图内的 `finish()`, 新增 `capture_stream`、`cuBLAS` 预热、`device index` 设置与 `abort_pending_run()`, 并让 `prepare()` 接受 `cg_mode / for_capture`。
- `vllm/v1/worker/gpu/dp_utils.py` (模块 数据并行同步; 类别 source; 类型 core-logic; 符号 `sync_cudagraph_and_dp_padding`): DP 同步是微批图分发的决策点: `sync_cudagraph_and_dp_padding` 从硬编码 `cg_mode=NONE` 改为向 `cudagraph manager` 请求微批 FULL 图, 并新增空第二微批守卫 (即合入后被确认存在边界 bug 的 $2 * \min < \text{num_tokens}$ 判断)。
- `tests/v1/worker/test_gpu_ubatch_slicing.py` (模块 微批测试; 类别 test; 类型 test-coverage; 符号 `test_capturable_run_replays_as_a_cudagraph`, `test_captured_split_survives_a_replay_with_enough_requests`, `test_microbatched_graphs_are_only_offered_to_uniform_batches`, `test_microbatched_graph_needs_every_rank_to_reach_the_split`): 新增 4 个针对性的 GPU 测试与 `_make_cudagraph_manager` 夹具, 覆盖回放读持久缓冲、切分在回放中保持、微批图只服务 uniform batch、以及任一 rank 达不到切分点时全组回退 eager。
- `vllm/v1/worker/gpu/model_runner.py` (模块 模型执行器; 类别 source; 类型 data-contract; 符号 `GPUModelRunner.initialize_kv_cache`, `GPUModelRunner.execute_model`): 接线改动: 把 `self.ubatch_runner` 传给 `ModelCudaGraphManager` (使候选生成可用), 并在 `prepare()` 调用处按 `batch_desc.cg_mode` 与 `dummy_run` 传入 `cg_mode / for_capture`。
- `vllm/config/vllm.py` (模块 配置校验; 类别 source; 类型 configuration; 符号 `VllmConfig._get_dbo_unsupported_features`): 配置校验放开: DBO 从“禁止任何 CUDA 图”改为只禁止 PIECEWISE, 是本次功能对用户可见的开关变化。

关键符号: `_maybe_ubatch_twin`, `_capture_stream`, `store_capture_output`, `ubatch_forward_fn`, `UBatchRunner.begin_capturable_run`, `UBatchRunner.abort_pending_run`, `UBatchRunner.prepare`, `sync_cudagraph_and_dp_padding`, `GPUModelRunner.execute_model`

关键源码片段

`vllm/v1/worker/gpu/cudagraph_utils.py`

本 PR 的核心: 新增微批候选生成 `_maybe_ubatch_twin`、捕获流选择 `_capture_stream`、`ubatch_forward_fn` 捕获分支与 `store_capture_output` 持久输出拷贝, 并让 `CudaGraphManager / ModelCudaGraphManager` 接收 `ubatch_runner`。

`vllm/v1/worker/gpu/cudagraph_utils.py` —— 生成微批孪生图候选, 并选定捕获流

```
def _maybe_ubatch_twin(
    self, desc: BatchExecutionDescriptor
) -> BatchExecutionDescriptor | None:
    """Return a microbatched capture candidate when eligible."""
```

```

# 只有启用 DBO 且是 FULL 模式时才可能生成微批图
if self.ubatch_runner is None or desc.cg_mode != CUDAGraphMode.FULL:
    return None
if desc.num_reqs is None:
    return None

# 能整除说明整个 batch 的 query 长度一致；只有这种形状才能在回放时
# 复现捕获时的 request 切分，否则切分比例会随 batch 漂移
uniform_token_count, remainder = divmod(desc.num_tokens, desc.num_reqs)
if remainder or desc.uniform_token_count not in (None, uniform_token_count):
    return None

parallel_config = self.vllm_config.parallel_config
num_ubatches = get_num_ubatches(parallel_config)
if desc.num_tokens < num_ubatches:
    return None
# 复用 DP 握手投票用的同一套阈值：候选生成必须是配置的纯函数，
# 否则会出现某 rank 回放图、另一 rank 走 eager，专家 all-to-all 直接挂住
if not check_ubatch_thresholds(
    parallel_config, desc.num_tokens, uniform_decode=True
):
    return None

return replace(
    desc, num_ubatches=num_ubatches, uniform_token_count=uniform_token_count
)

def _capture_stream(self, desc: BatchExecutionDescriptor) -> torch.cuda.Stream:
    """Capture on the stream used by the microbatch threads."""
    if desc.num_ubatches > 1:
        assert self.ubatch_runner is not None
        # 微批线程只把 kernel 发到 capture_stream；torch.cuda.graph 默认另开
        # side stream，不显式指定会把线程的工作留在图外，等于什么都没捕获
        return self.ubatch_runner.capture_stream
    return current_stream()

# capture() 中的事务边界：失败时用 abort_pending_run 释放被 park 的线程
with graph_capture(device=self.device), ExitStack() as stack:
    if self.ubatch_runner is not None:
        stack.callback(self.ubatch_runner.abort_pending_run)
    ...
    with torch.cuda.graph(graph, self.pool, stream=self._capture_stream(desc)):
        forward_fn(CUDAGraphMode.NONE)

```

vllm/v1/worker/gpu/ubatch_utils.py

把原来的 `run()`（线程创建 + barrier + join 一体化）拆成图外的 `begin_capturable_run()` 与图内的 `finish()`，新增 `capture_stream`、cuBLAS 预热、device index 设置与 `abort_pending_run()`，并让 `prepare()` 接受 `cg_mode / for_capture`。

vllm/v1/worker/gpu/ubatch_utils.py —— 图外 begin + 图内 finish 的两段式执行

```
def begin_capturable_run(
    self,
    model: Any,
    model_inputs: dict[str, Any],
    ubatch_state: UBatchState,
    for_capture: bool = False,
) -> Callable[[], Any]:
    """Start microbatch threads and return a callback to run and join them."""
    ubatch_slices = ubatch_state.slices
    assert len(ubatch_slices) == self.num_ubatches

    # 捕获时线程必须和 torch.cuda.graph 落在同一条流上
    compute_stream = self.capture_stream if for_capture else current_stream()
    ubatch_contexts = make_ubatch_contexts(
        num_micro_batches=self.num_ubatches,
        comm_stream=self.comm_stream,
        compute_stream=compute_stream,
        forward_contexts=ubatch_state.forward_contexts,
        ready_barrier=self.ready_barrier,
    )

    @torch.inference_mode()
    def run_ubatch(ubatch_context, inputs):
        try:
            # 新建线程不继承主线程的 device index，必须显式设置
            torch.accelerator.set_device_index(self.device.index)
            if for_capture:
                # cuBLAS 首次在某条流上使用才分配 workspace，捕获中途分配非法，
                # 所以提前把 compute_stream 与 comm_stream 都预热一遍
                for stream in (compute_stream, self.comm_stream):
                    with torch.cuda.stream(stream):
                        torch.cuda.current_blas_handle()
            with ubatch_context:
                outputs[ubatch_context.id] = model(**inputs)
        except BaseException as e: # noqa: BLE001
            errors[ubatch_context.id] = e

    # 线程与 ready barrier 留在图外，把它们 park 住后返回回调
    ...
    self.ready_barrier.wait()

    def finish() -> Any:
        # 图内只有事件置位与 join，没有任何逐微批的 Python 调度逻辑
```

```

try:
    ubatch_contexts[0].cpu_wait_event.set()
    for thread in threads:
        thread.join()
finally:
    self._pending_finish = None
    stack.close()
...

self._pending_finish = finish
return finish

```

```

def abort_pending_run(self) -> None:
    """Release and join parked threads after a failed capture."""
    if self._pending_finish is None:
        return
    try:
        self._pending_finish()
    except Exception:
        logger.exception("Failed to join microbatch threads after an abort")
    finally:
        self._pending_finish = None

```

vllm/v1/worker/gpu/dp_utils.py

DP 同步是微批图分发的决策点： `sync_cudagraph_and_dp_padding` 从硬编码 `cg_mode=NONE` 改为向 cudagraph manager 请求微批 FULL 图，并新增空第二微批守卫（即合入后被确认存在边界 bug 的 $2 * \min < \text{num_tokens}$ 判断）。

vllm/v1/worker/gpu/dp_utils.py —— 微批分支的图分发与守卫

```

if check_ubatch_thresholds(parallel_config, int(num_tokens_across_dp.min()),
                           uniform_decode=uniform_decode_across_dp):
    # 专家 all-to-all 要求所有 rank 等量切分并 pad 到相同 token 数,
    # 空微批按 dummy batch 运行
    ubatch_num_tokens = int(num_tokens_across_dp.max())
    num_ubatches = get_num_ubatches(parallel_config)
    ubatch_desc = None
    if cudagraph_manager is not None:
        # 用同步后的 uniform token count 请求微批 FULL 图,
        # dispatch 匹配不到时会回退成 NONE 描述符
        ubatch_desc = cudagraph_manager.dispatch(
            num_reqs,
            ubatch_num_tokens,
            synced_uniform_token_count,
            num_active_loras=num_active_loras,
            num_ubatches=num_ubatches,
        )
    # 若最小 rank 连第二个微批都填不满 (2 * min 覆盖不了图的 padded 大小),

```

```

# 则退回 eager, 避免用全 padding 的微批回放 FULL 图
if 2 * int(num_tokens_across_dp.min()) < ubatch_desc.num_tokens:
    ubatch_desc = None
if ubatch_desc is None:
    ubatch_desc = BatchExecutionDescriptor(
        cg_mode=CUDAGraphMode.NONE,
        num_tokens=ubatch_num_tokens,
        num_reqs=num_reqs,
        num_ubatches=num_ubatches,
    )
# 刷新 token 数, 把 CUDA 图 padding 也算进去
ubatch_num_tokens = ubatch_desc.num_tokens
num_reqs = int(num_reqs_across_dp.max())
if ubatch_desc.cg_mode == CUDAGraphMode.FULL:
    # FULL 图对请求数有固定要求, 必须用捕获时的 num_reqs
    assert ubatch_desc.num_reqs is not None
    num_reqs = ubatch_desc.num_reqs
return ubatch_desc, DPSyncState(
    num_tokens_across_dp=torch.full_like(num_tokens_across_dp, ubatch_num_tokens),
    uniform_token_count=synched_uniform_token_count,
    eager=ubatch_desc.cg_mode == CUDAGraphMode.NONE,
    num_reqs=num_reqs,
)

```

评论区精华

1) 注释风格收敛: SageMoore 在 `dp_utils.py` 上给出多条 suggestion, 要求把成段解释压成两行 (如“# Match a FULL microbatched CUDA graph, falling back to an eager descriptor if no graph matches.”), njhill 明确背书: “I agree with @SageMoore's comments. In general lets keep comments very concise.”; SageMoore 在 `model_runner.py` 上还有 “NIT: Let's just delete this comment.”, njhill 回复 “+1”。作者随后用 `a54a49b`、`882f2cc` 两个 commit 落实 “Simplify dispatch comments and padding updates, and keep capture and cleanup constraints concise”。2) FULL 图的请求数不变量: SageMoore 建议不要在分支里就地改写 `num_reqs`, 而是显式写成 `num_reqs = int(num_reqs_across_dp.max())` 加 `if ubatch_desc.cg_mode == FULL: assert ubatch_desc.num_reqs is not None; num_reqs = ubatch_desc.num_reqs`, 把 FULL 图对请求数的要求变成可读不变量, 已被采纳。3) 合入后暴露的边界缺陷: Laceprndpm 指出在 DP token 数为 `[64, 128]`、存在 128-token 图时, “the smaller rank's second microbatch is entirely padding, but $2 * 64 < 128$ still permits FULL replay”, 并准备了 `< → <=` 的补丁与回归参数; 作者回复 “I think you're right. This is a bug instead of intentional behavior. Thank you for pointing out, and please open a PR to fix it.”——这是目前唯一被确认的遗留正确性问题。4) 候选图可能永不命中: coderabbitai 的 nit 指出 mixed 分支中可被整除的捕获尺寸可能生成 `uniform_token_count > 1` 的孪生图 (例如 `512 / 256 == 2`), 而 DBO 已拒绝投机解码、运行时 query 长度为 1、`_is_compatible` 又要求精确匹配, 这些图捕获后无法被回放, 只会白占显存; 在可见讨论中未见回应或修复。5) 刻意排除的失败处理缺口: aoshen02 指出 `ubatching.py` 中兄弟微批出错挂死、以及 `__exit__` 清空

`_CURRENT_CONTEXTS[self.id]` 后唤醒 peer 导致 `AttributeError: 'NoneType' object has no attribute 'recv_hook'` 的偶发崩溃，已另开 #55586；本 PR 通过测试 docstring 记录该缺口但明确 out of scope。

- 注释冗长，要求大幅精简 (style): 作者在 a54a49b (Address DBO CUDA graph review nits) 与 882f2cc (Shorten DBO CUDA graph comments) 两个 commit 中落实，合并评论中确认 “Your suggestions are applied and I made the comments very concise.”
- FULL 图的请求数不变量显式化 (design): 已采纳，FULL 微批图对 num_reqs 的固定要求变成可读的显式契约，并在断言保护下生效。
- 空第二微批守卫的边界错误 (correctness): 作者确认: “I think you're right. This is a bug instead of intentional behavior. Thank you for pointing out, and please open a PR to fix it.”，修复留给 follow-up；这是本 PR 目前唯一被确认的遗留正确性缺陷。
- 部分微批孪生图永远不会被回放 (performance): 可见讨论中未见回应，问题保持开启；考虑到显存开销实测已从 0.46 升到 0.96 GiB，值得作为后续清理项跟踪。
- ubatching.py 微批失败处理与 sporadic 崩溃 (correctness): 本 PR 明确 out of scope (需改共享的 ubatching.py 交接协议)，已被另开 #55586 处理；作者在测试 docstring 中保留了对该缺口的说明。

风险与影响

- 风险：1) 空第二微批守卫边界 (已确认 bug) : `vllm/v1/worker/gpu/dp_utils.py` 中 `2 * int(num_tokens_across_dp.min()) < ubatch_desc.num_tokens` 未覆盖相等情形，DP 负载极不均衡 (如 [64, 128] 对 128-token 图) 时仍会用实际内容全为 padding 的微批去回放 FULL 图，属于合入后遗留的正确性缺口，需按 `Laceprndpm` 的补丁把 `<` 改为 `<=` 并补回归参数。2) 跨 rank 候选集合一致性靠约定维持: `_maybe_ubatch_twin` 必须与 `UBatchRunner.wants_ubatch`、DP 握手投票使用同一判据 (`check_ubatch_thresholds + get_num_ubatches`)，一旦有人只改一侧，就会出现“某 rank 回放、另一 rank eager”的不对称执行，专家 all-to-all 会挂死；当前没有运行时断言兜底。3) 捕获路径的两条隐性约束易被回归: 捕获必须落在微批线程的 `capture_stream` 上，且线程需在捕获前完成 device index 设置与 cuBLAS workspace 预热——PR body 明确说这两点是让 draft “捕获不到任何东西”的原因；后续重构 `begin_capturable_run` 或改动 `make_ubatch_contexts` 的流参数时容易再次踩坑。4) 持久输出缓冲的首次分配时机: `store_capture_output` 在 `hidden_states / aux_hidden_states / intermediate_tensors` 为 `None` 时使用 `torch.empty_like` 分配，因此依赖捕获前的 `warmup` 调用顺序；若 `create_forward_fn(..., warmup=...)` 的调用序列被调整，分配将落入捕获区而报错。5) 显存与配置行为变化: `config/vllm.py` 放开 DBO + FULL 后，此前在配置校验阶段被拒绝的组合会真正为每个 FULL 候选多捕获一张微批图，Qwen3-30B-A3B 实测捕获开销从 0.46 GiB 增至 0.96 GiB，小显存或捕获尺寸密集的部署需要重新评估；PIECEWISE 仍然被拒绝，用户若用 PIECEWISE 会得到与以前不同的报错文案。6) 验证强度有限: 精度证据是 GSM8K 单次运行 (run-to-run 波动约 1pp)，DeepSeek-V2-Lite 上 0.3503 与 no-DBO 0.3457 的差异落在噪声范围内；作者自述 “Perf validation is deferred to P4 and needs cross-node EP”，本 PR 未提供吞吐量数据，FULL 图带来的收益目前只有结构性论证与“分支确实被回放”的日志证据。

- 影响：对用户：使用 `VLLM_USE_V2_MODEL_RUNNER=1` 的 DP+EP MoE 部署（DeepSeek-V2-Lite、Qwen3-30B-A3B-Thinking-2507-FP8 等）现在可以开启 `--enable-dbo` 并走 FULL CUDA 图回放，decode 步的 CPU 发射路径与微批交接开销从热路径消失；只支持 uniform batch 的 MLA 后端在 mixed batch 上仍按设计回退 eager 微批，因此收益是按后端与负载形态分层的。显存占用会因多捕获一组微批图而上升，需要重新核算。对系统：BatchExecutionDescriptor 的 `num_ubatches` 从“仅是描述字段”变成了图选择的关键键（`_is_compatible` 要求精确匹配），MRV2 的 dispatch 契约、捕获契约（图外准备 + 图内 finish）、以及 DP 同步契约（同步后再 dispatch）都被本次改动同时收紧，后续所有涉及 cudagraph 的改动都要考虑这一维度。对团队：MRV2/DBO 主线推进到 P4 前夜，DP+EP 迁移 V2 的最后一个已知阻塞被移除；协作上呈现“作者 + 社区多轮迭代（5 个 commit 中 2 个是 review 后的注释精简、2 个是 rebase）”，并明确把边界修复与跨节点性能验证留给 follow-up。
- 风险标记：核心捕获路径变更，合并后遗留边界 bug，多 rank 一致性依赖约定，显存占用上升，缺少端到端性能验证

关联脉络

- PR #50945 [1/2][Model Runner V2] DBO support, eager mode (P0-P2): 本 PR 的直接前置与 stacked base: 本 PR 在其之上补 FULL 图捕获，PR body 明确“Stacked on top of #50945”。
- PR #55586 ubatching.py 微批失败 / 挂死修复（由 aoshen02 提及）：讨论中由 aoshen02 针对本 PR 测试 docstring 记录的失败处理缺口提起，修复同一 DBO 交接协议中的挂死与 `NoneType.recv_hook` 崩溃。
- PR #53867 [Feature][PCP] Support decode-only FULL CUDA graphs: 同属 Model Runner V2 worker 的 FULL CUDA 图工作线，改动同样落在 `vllm/v1/worker/gpu/cudagraph_utils.py`、`model_runner.py` 与 `input_batch.py`，本 PR 的捕获流与候选生成逻辑与该线共享同一套 dispatch 机制。