PR #34464 完整报告

sgl-project/sglang

Refocus LoRA tests on regression coverage

合并时间：2026-08-12 08:37

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34464

执行摘要

- 一句话：LoRA 测试聚焦回归契约，方法数削减 61%
- 推荐动作：值得精读，推荐对象是对测试治理、回归守护和 LoRA 特性感兴趣的同学。重点看三点：一是 test_chunked_sgmv_cuda_graph.py 如何用「捕获时 1 段、重放时 4 段」的状态切换构造 CUDA graph 动态 segment 回归，技巧可复用于其他 CUDA graph 类测试；二是 test_lora_manager_tied_lm_head.py 用 patch.object + patch.dict(sys.modules) 在 CPU 上复现 GPU/ 模型加载路径的回归，极大降低成本；三是 PR body 的 admission 决策表是测试裁剪的范本，可迁移到其他测试套件。若你负责 LoRA 相关改动，建议后续改动时先跑这些契约验证。

功能与动机

PR body 明确指出：Apply .claude/rules/unit-test-admission.md to the LoRA tests so each retained case names a concrete future regression。背景是 #34070 的 broad cleanup 把一批有价值的回归测试与冗余覆盖一起删掉了，本 PR 在保留 admission 规则的前提下恢复「只聚焦契约束」的覆盖，并修剪无法证明具体回归的弱 happy path（例如 test_lora_radix_cache.py 的旧断言不证明 radix 命中）。

实现拆解

1. 建立 admission 基线：对照 #34070 之前的 LoRA suite，逐一给每个文件标注 before/after/ 保留理由 /CI 归属，作为本次裁剪的决策表。
2. 恢复并重写 CUDA graph 回归（test/registered/kernels/ops/gemm/test_chunked_sgmv_cuda_graph.py，16 方法 → 5 契约）：保留并重构 #28371 的四类 CUDA graph 回归——shrink/expand 重放必须使用当前全部 segment、prepare_lora_batch 必须中和静态尾部 segment、以及 MLA 吸收后 KV 分支的 step_a_q_fwd/step_b_q_fwd/step_a_v_fwd/step_b_v_fwd 重放回归；删除了在其他文件已有覆盖的通用 kernel sweep。
3. 驱逐策略测试 CPU 化：删除 test/registered/lora/test_lora_eviction_policy.py（201 行，原注册 CUDA/AMD/XPU/base-c），新建 test/registered/unit/lora/test_eviction_policy.py（5 个契约，注册 base-a-test-cpu，1s），保留 LRU-vs-FIFO 语义、候选过滤、remove 记账、非法策略拒绝、#14795 base-model-last 回归。
4. OpenAI 兼容测试精简为负向契约（test/registered/lora/test_lora_openai_compatible.py，11 方法 → 2 方法）：只保留「未知 adapter 必须被拒绝而非静默落到 base model」和「LoRA 禁用时请求必须报错并提示 --enable-lora」两个负向路由契约；移除了失去 LoRA

仍保持绿色、无法识别具体回归的 happy path，并删除 AMD 注册与外部 adapter 下载依赖。

5. 新增 / 迁移其余聚焦契约：test_lora_manager_tied_lm_head.py 用 mock/patch 在 CPU 上直接构造 #18634 tied-lm_head 去重回归，替换原 1 个重型 GPU E2E 测试；test_lm_head_pruning.py 在 CPU 上守护 LogitsProcessor._get_pruned_states 与 get_lm_head_pruned_lens 的逐请求行数一致；test_io_struct.py 新增 embedding 请求批拆分后 lora_path/lora_id 存活与数量校验两个契约；test_embedding_lora_support.py 重建为「base vs LoRA 非平凡 + HF parity oracle」的单点 E2E 回归（nightly）。
6. CI/ 配置配套：test_lora_radix_cache.py 保持已删除状态；CUDA 图回归注册到 base-b-kernel-unit（1-gpu-large），embedding oracle 与 OpenAI 负向契约保留 nightly CUDA 注册，其余纯 Python 覆盖统一收敛到 CPU base-a。

关键文件：

- test/registered/kernels/ops/gemm/test_chunked_sgmv_cuda_graph.py（模块 CUDA 图；类别 test；类型 test-coverage；符号 _make_batch_info, _set_segment_state, _capture, test_shrink_replay_uses_all_current_segments）：本次变更中权重最高的回归测试：把 #28371 的 CUDA graph 动态 LoRA segment 回归精炼为 5 个聚焦契约，覆盖 shrink/expand 重放、prepare_lora_batch 尾部中和、MLA 吸收后 KV 分支重放，是 LoRA 在 CUDA graph 路径上的核心守护。
- test/registered/lora/test_embedding_lora_support.py（模块 嵌入支持；类别 test；类型 test-coverage；符号 TestEmbeddingLoRAParity, _hf_embeddings, _sglang_embeddings, extract_embeddings）：重建为端到端 embedding LoRA 的独立 HF parity oracle：先验证 LoRA 对 embedding 有非平凡影响（base vs LoRA 不相等），再与 Hugging Face peft 输出做相似度对比，替代原先无效的镜像断言。
- test/registered/lora/test_lora_eviction_policy.py（模块 驱逐策略；类别 test；类型 deletion；符号 TestLoRAEvictionPolicy, _test_eviction_policy, test_lru_basic, test_lru_with_reuse）：旧的 12 方法策略测试整体删除，迁移并收敛为 test/registered/unit/lora/test_eviction_policy.py 的 5 个 CPU 契约，同时移除 CUDA/AMD/XPU/base-c 四处注册。
- test/registered/unit/lora/test_eviction_policy.py（模块 驱逐策略；类别 test；类型 test-coverage；符号 TestLoRAEvictionPolicy, _make_policy, test_reuse_changes_lru_but_not_fifo, test_selection_skips_older_noncandidates）：驱逐策略测试的新家：纯 Python 覆盖迁移到 CPU base-a（1 秒），保留 LRU/FIFO 语义差异、候选过滤、remove 记账、非法策略拒绝和 #14795 base-model-last 回归，是测试资源优化最典型的文件。
- test/registered/lora/test_lora_openai_compatible.py（模块 API 路由；类别 test；类型 test-coverage；符号 setup_class, TestLoRAOpenAICompatible, test_unknown_model_adapter_is_rejected, TestLoRADisabledError）：从 11 个 happy path 精简约到 2 个负向路由契约，移除 AMD 注册与真实 adapter 下载依赖，避免"LoRA 被静默丢弃时仍保持绿色"的无效覆盖。
- test/registered/unit/lora/test_lora_manager_tied_lm_head.py（模块 LoRA 管理器；类别 test；类型 test-coverage；符号 _TiedEmbedding, _ParallelLMHead,

TestTiedLMHeadLoRA, test_tied_head_gets_independent_wrapper_with_shared_base_weight）：用 mock/patch 在 CPU 上直接复现 #18634 tied lm_head 去重故障，替换原1个需要 GPU 与模型下载的重型 E2E 测试，是“重型测试 CPU 化”的代表作。

- test/registered/unit/lora/test_lm_head_pruning.py（模块 头部裁剪；类别 test；类型 test-coverage；符号 TestLMHeadPruning, test_lora_segments_match_pruned_rows_per_request）：新增契约守护 LoRA lm_head 分段与 logits state 裁剪的逐请求行数一致，防止“总数相等但路由错位”被测试放过的场景。
- test/registered/unit/managers/test_io_struct.py（模块 请求结构；类别 test；类型 test-coverage；符号 test_lora_identity_survives_batch_split, test_lora_path_count_must_match_embedding_batch）：补充 embedding 批量请求的 LoRA 结构契约：批拆分后 lora_path/lora_id 必须随子请求存活，且 lora_path 数量与 batch 不匹配时必须报错而非静默路由到 base。

关键符号：test_shrink_replay_uses_all_current_segments, test_expand_replay_uses_all_current_segments, test_prepare_batch_neutralizes_static_tail_segments, test_absorbed_kv_b_replay_uses_all_current_segments, test_hf_sglang_embedding_similarity, test_unknown_model_adapter_is_rejected, test_lora_disabled_error, test_tied_head_gets_independent_wrapper_with_shared_base_weight, test_reuse_changes_lru_but_not_fifo, test_selection_skips_older_noncandidates, test_base_model_is_evicted_only_as_last_resort, test_remove_excludes_adapter_from_future_selection, test_unknown_policy_is_rejected, test_lora_segments_match_pruned_rows_per_request, test_lora_identity_survives_batch_split, test_lora_path_count_must_match_embedding_batch, _set_segment_state, _capture

关键源码片段

test/registered/kernels/ops/gemm/test_chunked_sgmv_cuda_graph.py

本次变更中权重最高的回归测试：把 #28371 的 CUDA graph 动态 LoRA segment 回归精炼为 5 个聚焦契约，覆盖 shrink/expand 重放、prepare_lora_batch 尾部中和、MLA 吸收后 KV 分支重放，是 LoRA 在 CUDA graph 路径上的核心守护。

```
"""CUDA-graph 回归测试的核心构造：用「活跃/非活跃」两种 segment 状态切换来验证
CUDA graph 重放时能正确使用当前 batch 的全部 LoRA segment（对应 #28371）。"""


def _set_segment_state(batch_info, *, active):
    """在 LoRABatchInfo 中切换 LoRA segment 状态。
    active=True 时构造 4 个 adapter segment（rank = 8），
    active=False 时退化为单 segment（rank = 0），
    用于模拟「捕获时为 1 段、重放时为 4 段」的动态变化。"""
    if active:
        lora_ranks = [MAX_RANK] * NUM_LORAS
        weight_indices = [1, 2, 3, 4]
        seg_indptr = [0, 2, 4, 6, BS]
```

```
    else:
        lora_ranks = [0] * NUM_LORAS
        weight_indices = [0]
        seg_indptr = [0, BS]

    num_segments = len(weight_indices)
    batch_info.lora_ranks.copy_(torch.tensor(lora_ranks, dtype=torch.int32, device="cuda"))
    batch_info.weight_indices.zero_()
    # 只填充前 num_segments 个有效槽位，其余保持为 0
    batch_info.weight_indices[:num_segments].copy_(
        torch.tensor(weight_indices, dtype=torch.int32, device="cuda")
    )
    batch_info.seg_indptr.fill_(seg_indptr[-1])
    batch_info.seg_indptr[: num_segments + 1].copy_(
        torch.tensor(seg_indptr, dtype=torch.int32, device="cuda")
    )
    batch_info.num_segments = num_segments


def _capture(call):
    """在独立 warmup stream 上预热 3 次后捕获 CUDA graph，返回 (graph, output)。
    预热必不可少：kernel 首次运行会触发 autotune，直接捕获可能把调优路径固化进图。"""
    warmup_stream = torch.cuda.Stream()
    warmup_stream.wait_stream(torch.cuda.current_stream())
    with torch.cuda.stream(warmup_stream):
        for _ in range(3):
            call()
    torch.cuda.current_stream().wait_stream(warmup_stream)
    torch.cuda.synchronize()

    graph = torch.cuda.CUDAGraph()
    with torch.cuda.graph(graph):
        output = call()
    return graph, output


def test_shrink_replay_uses_all_current_segments():
    """一个「单 segment 捕获」的图，在重放 4 个 adapter 后必须与 eager shrink 一致。"""
    _chunked_lora_shrink_kernel._clear_cache()
    batch_info = _make_batch_info()
    inputs = torch.randn(BS, 64, dtype=torch.float16, device="cuda")
    weights = torch.randn(NUM_LORAS, MAX_RANK, 64, dtype=torch.float16, device="cuda")

    # 先以 4-segment 的 eager 结果作为基准
    _set_segment_state(batch_info, active=True)
    expected = chunked_sgmv_lora_shrink_forward(
        inputs, weights, batch_info, num_slices=1
    ).clone()
```

```
    # 以 1-segment 状态捕获图：重放时必须重新读取 weight_indices / seg_indptr
    _set_segment_state(batch_info, active=False)
    graph, captured_output = _capture(
        lambda: chunked_sgmv_lora_shrink_forward(inputs, weights, batch_info, num_slices=1)
    )

    # 恢复 4-segment 状态并重放，输出必须与 eager 基准一致
    _set_segment_state(batch_info, active=True)
    captured_output.zero_()
    graph.replay()
    torch.cuda.synchronize()

    torch.testing.assert_close(
        captured_output[:, :MAX_RANK],
        expected[:, :MAX_RANK],
        rtol=RTOL,
        atol=ATOL,
    )
```

test/registered/unit/lora/test_lora_manager_tied_lm_head.py

用 mock/patch 在 CPU 上直接复现 #18634 tied lm_head 去重故障，替换原 1 个需要 GPU 与模型下载的重型 E2E 测试，是“重型测试 CPU 化”的代表作。

```
"""回归 #18634：LoRA 包装与输入输出 embedding 共享对象的 lm_head 时的去重行为。
整个测试在 CPU 上运行（base-a，1 秒），通过 patch 替代 GPU 与模型下载。"""

class TestTiedLMHeadLoRA(CustomTestCase):
    def test_tied_head_gets_independent_wrapper_with_shared_base_weight(self):
        """仅挂载 lm_head 的 LoRA 必须能在 tied 权重下存活：
        lm_head 被换成独立包装器，但包装器内部仍与 embed_tokens 共享参数字面量。"""
        model = torch.nn.Module()
        tied_embedding = _TiedEmbedding()
        model.embed_tokens = tied_embedding
        model.lm_head = tied_embedding # 模拟 tied input/output embedding

        manager = LoRAManager.__new__(LoRAManager)
        manager.base_model = model
        manager.base_hf_config = SimpleNamespace(num_hidden_layers=0)
        manager.target_modules = {"lm_head"}
        wrapped_lm_head = object()

        # guard：避免真实 import 拉入大量 CUDA 依赖，仅做模块级替换
        inkling_module = types.ModuleType("sglang.srt.models.inkling_common.dense_mlp")
        inkling_module.InklingBatchDenseMLP = type("InklingBatchDenseMLP", (), {})

        with (
            patch.object(lora_manager_module, "ParallelLMHead", _ParallelLMHead),
            patch.object(manager, "set_lora_module", return_value=wrapped_lm_head) as set_lora_
            module,
```

```
            patch.dict(sys.modules, {"sglang.srt.models.inkling_common.dense_mlp": inkling_
            module}),
        ):
            manager.init_lora_modules()

        # lm_head 不再与 embed_tokens 是同一个对象，但两者共享底层 weight
        self.assertIsNot(model.lm_head, model.embed_tokens)
        self.assertIs(model.lm_head.weight, model.embed_tokens.weight)
        self.assertIs(manager.lm_head_module, wrapped_lm_head)
        set_lora_module.assert_called_once_with("lm_head", model.lm_head)
```

评论区精华

该 PR 没有产生实质 review 评论（review_comments_count = 0），合并前仅有一次 /rerun-test 触发 CI 重跑，两个 issue 评论都是命令与结果的自动化回执。真正值得关注的设计讨论全部沉淀在 PR body 的逐测试 admission 决策表里：例如对 test_lora_openai_compatible.py 明确写出「happy paths that stayed green when LoRA was dropped are removed」，即承认这批测试在 LoRA 被静默丢弃时仍然通过、属于无效守护；对 test_lora_radix_cache.py 则判定「The old assertion did not prove a radix-cache hit and did not identify a distinct future regression」。这些决策体现了「测试必须能指认具体回归」的取舍原则，是本次变更的方法论核心。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 大面积删除既有覆盖：28 个方法被移除（包含 OpenAI 兼容的全部 happy path、LRU/FIFO 的多数组合场景、AMD/XPU 的注册）。虽然每个删除都给出了 admission 理由，但若这些 happy path 曾隐式守护过某些非预期行为，保护会丢失；特别是 test_lora_eviction_policy.py 删除后，驱逐策略在 CUDA/AMD/XPU 平台上的行为不再有直接覆盖。
  2. 新增 CUDA graph 测试的 flaky 风险：test_chunked_sgmv_cuda_graph.py 涉及独立 warmup stream、CUDA graph 捕获与重放、_clear_cache() 与 torch.cuda.synchronize()，在 1-gpu-large 上运行，stream 时序敏感，存在偶发失败可能（PR 内已有一次 /rerun-test）。
  3. 外部依赖：test_embedding_lora_support.py 需要从 Hugging Face 下载 meta-llama/Llama-2-7b-hf 与 LoRA adapter，且要动态加载 peft/transformers，对网络与显存要求高（nightly 150s），存在外部源不可用或版本漂移导致误报的风险。
  4. 覆盖转移而非新增：test_lora_radix_cache.py 保持删除，意味着「LoRA 与 radix cache 命中」这一组合路径在当前测试体系中没有任何守卫，后续若该交互回归将无法被 CI 捕捉。
  5. 无任何源码 / 配置改动，不影响线上推理路径，回归风险仅限测试面。 - 影响：对 用户：零运行时影响，模型输出、API 行为、性能均不变。对 CI 系统：净减少约 28 个测试方法与 4 类跨平台注册（AMD、XPU、base-c、nightly CUDA 中的部分项），驱逐策

略与 tied-lm-head 从 GPU/ 模型下载依赖降为 CPU 1 秒测试，CI 资源消耗显著下降。对 团队：这是一次测试治理范式的落地实践——以 .claude/rules/unit-test-admission.md 为标尺，确立了「每个用例必须对应一个可指认的回归」的评审标准，对后续新测试有直接示范作用。对 LoRA 功能线：回归保护更集中但覆盖范围变窄，未来修改eviction policy、OpenAI 路由、CUDA graph 段逻辑时需要依赖这批聚焦契约。影响程度：中等，属于工程效率与质量保障层面的改进。 - 风险标记：大面积删除既有测试覆盖，新增 CUDA 图测试 flaky 风险，外部模型下载依赖，跨平台注册移除

关联脉络

- PR #34070 （标题未在材料中提供）：PR body 明确引用该 PR：其 broad cleanup 删除了大量冗余覆盖的同时误删了若干有价值的 LoRA 回归测试，本 PR 正是在 admission 规则下恢复其中值得保留的聚焦契约。