PR #32393 完整报告

sgl-project/sglang

[Spec] Share the grammar mask build and verify-tree staging across spec workers

合并时间：2026-07-25 18:40

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/32393

执行摘要

- 一句话：共享 spec worker 间的 grammar mask 构建与 verify-tree staging
- 推荐动作：建议仔细阅读 spec_utils.py 中新增的 GrammarTree 类和 build_grammar_vocab_mask 函数，尤其是在异步 GPU/CPU 交互场景下如何安全 staging 数据。该 PR 展示了将跨 worker 的通用模式提取为工具的最佳实践，值得在团队内推广。PR 讨论中关于设计窄接口的理由也值得学习。

功能与动机

每个 worker 重复实现 generate_token_bitmask -> .to(device, non_blocking=True) -> clear sampling_info.vocab_mask，容易出错且不易维护。两个即将加入 grammar 支持的 worker (#30096, #31753) 如果继续各自实现会加剧重复和风险。通过提取公共工具，新 worker 只需提供自己的 tree 即可。

实现拆解

1. 新增 GrammarTree 类（spec_utils.py）：封装三个 verify-tree tensor（retrieve_next_token, retrieve_next_sibling, draft_token）的异步 D2H 复制，提供 from_device（发出异步复制并记录 event）和 from_host（直接包装 host tensor）工厂方法，resolve 方法在首次访问时同步等待 event。
2. 新增 build_grammar_vocab_mask 函数（spec_utils.py）：接收 GrammarTree 并调用 resolve 获取 host tensor，然后调用已有的 generate_token_bitmask 构建 bitmask，再以 non_blocking=True 上传到 device，同时清除 sampling_info.vocab_mask 避免残留 extend-stage 的 mask。
3. 改造 EAGLE worker（eagle_worker_common.py）：将原本的三次 _async_d2h + event 记录替换为一行 GrammarTree.from_device(...)，并将 grammar_copy_done.synchronize() + generate_token_bitmask + 上传逻辑替换为 build_grammar_vocab_mask。
4. 改造 NGRAM worker（ngram_worker.py）：将原本的 generate_token_bitmask + vocab_mask.to(device, non_blocking=True) + vocab_mask = None 替换为 build_grammar_vocab_mask，传入 GrammarTree.from_host 因为 tree 已经在 host 上。
5. 保留 device tree links：eagle_sample 和 mamba/gdn/kda 后端仍然直接读取 device tensor，只将 grammar mask 构建路径抽离。

关键文件：

- python/sglang/srt/speculative/spec_utils.py（模块 推测工具；类别 source；类型 core-logic；符号 GrammarTree, init, from_device, from_host）：新增 GrammarTree 类和 build_grammar_vocab_mask 函数，是核心重构的载体。所有共享逻辑集中于此。
- python/sglang/srt/speculative/eagle_worker_common.py（模块 EAGLE 工作器；类别 source；类型 dependency-wiring）：移除手工 _async_d2h + event 同步，改为使用 GrammarTree.from_device 和 build_grammar_vocab_mask，减少 41 行重复代码。
- python/sglang/srt/speculative/ngram_worker.py（模块 NGRAM 工作器；类别 source；类型 core-logic）：使用 GrammarTree.from_host 和 build_grammar_vocab_mask 简化 NGRAM 的 grammar mask 构建，消除重复的 to(device) 和 vocab_mask 清理。

关键符号：GrammarTree.init, GrammarTree.from_device, GrammarTree.from_host, GrammarTree.resolve, build_grammar_vocab_mask

关键源码片段

python/sglang/srt/speculative/spec_utils.py

新增 GrammarTree 类和 build_grammar_vocab_mask 函数，是核心重构的载体。所有共享逻辑集中于此。

```
class GrammarTree:
    """The verify tree the grammar bitmask is built over, on the host.
    ``from_device`` starts an async copy, so build it before the target verify
    launch; ``from_host`` is for algorithms that build the tree there (NGRAM).
    """

    def __init__(self, host: Tuple[torch.Tensor, ...], done_event):
        self._host = host
        self._done = done_event

    @classmethod
    def from_device(
        cls,
        retrieve_next_token: torch.Tensor,
        retrieve_next_sibling: torch.Tensor,
        draft_token: torch.Tensor,
    ) -> "GrammarTree":
        tensors = (retrieve_next_token, retrieve_next_sibling, draft_token)
        host = tuple(_async_d2h(t) for t in tensors) # 异步 D2H，不阻塞当前流
        # Sources may be mixed -- an algorithm can synthesize part of the tree on
        # the host -- so the event has to key off whichever one is on device.
        device = next((t.device for t in tensors if t.device.type != "cpu"), None)
        if device is None:
            return cls(host, None)
        done = torch.get_device_module(device).Event()
        done.record() # 在复制完成后记录 event
        return cls(host, done)
```

```
    @classmethod
    def from_host(
        cls,
        retrieve_next_token: torch.Tensor,
        retrieve_next_sibling: torch.Tensor,
        draft_token: torch.Tensor,
    ) -> "GrammarTree":
        # 直接包装 host tensor，无需 event
        return cls((retrieve_next_token, retrieve_next_sibling, draft_token), None)

    def resolve(self) -> Tuple[torch.Tensor, ...]:
        # 需要同步时等待 D2H 完成
        if self._done is not None:
            self._done.synchronize()
        return self._host


def build_grammar_vocab_mask(
    *,
    reqs: List[Req],
    verify_input: SpecInput,
    tree: GrammarTree,
    sampling_info: SamplingBatchInfo,
    device,
) -> Optional[torch.Tensor]:
    """Build the constrained-decoding bitmask over a verify tree and stage it on device.
    Call it after the target verify launch: resolving the tree and traversing it are
    both host work, so both overlap that forward.
    """
    vocab_mask = generate_token_bitmask(
        reqs,
        verify_input,
        *tree.resolve(), # 同步等待 D2H 完成（如果需要），然后遍历构建 mask
        sampling_info.vocab_size,
    )
    if vocab_mask is None:
        return None

    assert verify_input.grammar is not None
    # non_blocking is safe: the bitmask is pinned (see xgrammar_backend), and stream
    # order keeps the copy ahead of the sampler's apply_vocab_mask.
    vocab_mask = vocab_mask.to(device, non_blocking=True)
    # Clear stale extend-stage mask before sampling.
    sampling_info.vocab_mask = None
    return vocab_mask
```

评论区精华

PR body 中 'Notes for reviewers' 部分阐述了关键设计权衡：

- GrammarTree 的窄设计：故意在当前 stream 上复制而不 clone，因为 tree 复制必须保持顺序在 draft kernel 之后、verify launch 之前。通用侧 stream 变体（如 kv_canary 的 FutureTensors）的 per-tensor clone 和专用 stream 不适合 per-step verify 路径。
- NGRAM 不 staging：NGRAM 的 tree 本身就在 host 上派生（#32380），不需要 D2H staging，因此直接调用 from_host，这证明了拆分 from_device / from_host 的实际价值。
- device tree links 保留：eagle_sample 等仍需 device 上的原始 tensor，因此 GrammarTree 只处理 grammar mask 构建所需的部分。
  - StagedGrammarTree 的设计范围 (design): 接受窄设计，因为验证步长所需语义恰好是顺序保持。
  - NGRAM 不 staging 的原因 (design): 设计允许不同 worker 选择合适的方式创建 GrammarTree，灵活且无额外开销。

风险与影响

- 风险：主要风险在于新代码引入的 bug 可能导致 grammar 约束失效或性能退化。但由于：
  1. PR 展示了完整的准确性测试结果（NGRAM JSON-schema decode 各项指标与重构前一致）；
  2. 替换是等价的——build_grammar_vocab_mask 内部调用 generate_token_bitmask 并做了同样的 non_blocking 上传和 vocab_mask 清理；
  3. 对非 grammar 路径完全无影响；
  4. 测试包括 NGRAM 和 EAGLE 约束解码，均通过。 因此风险较低。潜在风险为 GrammarTree.resolve 中 event 同步失败导致 host 读取未就绪数据，但代码已针对 device 类型做了检查，且 event 在异步复制后立即记录，同步发生在遍历前，顺序正确。
- 影响：范围：仅影响 speculative decoding worker 内部，对用户无感知（no behavior change）。程度：中等到高，因为消除了重复代码并统一了实现，使未来更多 worker 添加 grammar 支持变得更简单可靠。性能：无退化，反而通过集中管理 non_blocking 上传减少出错可能，间接有利于性能。可维护性：显著提升，减少了 EAGLE 和 NGRAM 中约 60 行重复 / 易错代码。
- 风险标记：核心路径变更，异步复制顺序敏感，已通过准确性测试

关联脉络

- PR #32380 [Spec] Derive NGRAM grammar tree links on the host instead of reading back retrive_next_token: 此 PR 将 NGRAM 的 tree 链接派生移到 host，使得 NGRAM worker 可以使用 GrammarTree.from_host，是本 PR 重构的前提之一。
- PR #32353 [Spec] Consolidate the grammar sync decision into ScheduleBatch.grammar_needs_sync: 同为 speculative grammar 相关重构，集中 grammar 同步判断，与本 PR 共享相同模块和背景。