PR #26195 完整报告

sgl-project/sglang

Allow custom speculative algorithm to support disaggregation

合并时间：2026-05-28 00:54

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26195

执行摘要

- 一句话：为自定义推测算法添加分离支持，重构 Eagle 专用逻辑
- 推荐动作：值得精读，展示了通过面向对象多态方法解耦调度逻辑的设计模式。建议尽快补充：1）空批次和 hidden states 为 None 的防御性检查；2）针对新增接口编写单元测试，覆盖 Eagle 和非 Eagle 分支的分离场景。

功能与动机

PR body 说明："When prefill/decode disaggregation is used with custom speculative algorithm, SpecInfo has to be populated in order for it to work correctly. Currently, it specializes for eagle only." 本 PR 旨在解除这个限制，使自定义推测算法也能获得分离解码支持。

实现拆解

1. 创建新文件 python/sglang/srt/speculative/eagle_disaggregation.py，将原先嵌入在 decode_schedule_batch_mixin 中的 Eagle 专用 draft 输入构建逻辑抽取为独立函数 build_eagle_disagg_draft_input()。该函数负责收集每个请求的 output_topk_p、output_topk_index 及 hidden_states_tensor，组装成 EagleDraftInput，并在启用 overlap 调度时发布与暂存 future 信息。
2. 在 SpeculativeAlgorithm 枚举类（spec_info.py）中新增 build_disagg_draft_input() 方法。当算法为 Eagle 时，委托调用 build_eagle_disagg_draft_input；否则返回 None。这为所有内置算法提供了统一入口，方便后续扩展。
3. 在自定义算法基类 CustomSpecAlgo（spec_registry.py）中添加相同的 build_disagg_draft_input() 方法，默认返回 None。插件可通过覆盖该方法来获得分离支持，无需改动调度器。
4. 修改 decode_schedule_batch_mixin.py 中的 process_prebuilt() 方法：删除原先 46 行嵌入的 Eagle 构造代码，替换为一行对 self.spec_algorithm.build_disagg_draft_input() 的调用。根据返回值决定是否设置 self.spec_info，若为 None 则使用 last_tokens_tensor 直接作为 decode 输入。同时调整 import，移除 CaptureHiddenMode。

关键文件：

- python/sglang/srt/speculative/eagle_disaggregation.py（模块 推测解码；类别 source；类型 core-logic；符号 build_eagle_disagg_draft_input）：核心新增文件，提取 Eagle 专

用 draft 输入构造逻辑为独立函数，是本次重构的关键

- python/sglang/srt/disaggregation/decode_schedule_batch_mixin.py（模块 分离调度；类别 source；类型 dependency-wiring）：主要修改点，将硬编码的 Eagle 构造代码替换为多态调用，大幅减少代码并提高扩展性
- python/sglang/srt/speculative/spec_info.py（模块 推测解码；类别 source；类型 core-logic；符号 build_disagg_draft_input）：在 SpeculativeAlgorithm 枚举中添加 build_disagg_draft_input 方法，提供统一接口
- python/sglang/srt/speculative/spec_registry.py（模块 推测解码；类别 source；类型 core-logic；符号 build_disagg_draft_input）：在 CustomSpecAlgo 基类中添加默认 build_disagg_draft_input 方法，确保自定义算法可扩展

关键符号：build_eagle_disagg_draft_input, build_disagg_draft_input

关键源码片段

python/sglang/srt/speculative/eagle_disaggregation.py

核心新增文件，提取 Eagle 专用 draft 输入构造逻辑为独立函数，是本次重构的关键

```
from __future__ import annotations

from typing import TYPE_CHECKING

import torch

from sglang.srt.model_executor.forward_batch_info import CaptureHiddenMode
from sglang.srt.speculative.eagle_info import EagleDraftInput

if TYPE_CHECKING:
    from sglang.srt.managers.overlap_utils import FutureMap
    from sglang.srt.managers.schedule_batch import ScheduleBatch
    from sglang.srt.server_args import ServerArgs


def build_eagle_disagg_draft_input(
    batch: ScheduleBatch,
    server_args: ServerArgs,
    last_tokens_tensor: torch.Tensor,
    future_map: FutureMap,
) -> EagleDraftInput:
    # 从 server_args 获取 topk 数，若启用多层 Eagle 则乘以步数
    num_states = server_args.speculative_eagle_topk
    if server_args.enable_multi_layer_eagle:
        num_states *= server_args.speculative_num_steps

    # 收集每个请求的 topk 概率张量，并堆叠成 batch 维度
    topk_p = torch.stack(
        [
            torch.as_tensor(
```

```
            req.output_topk_p[:num_states],
            device=batch.device,
            dtype=torch.float32,
        )
        for req in batch.reqs
    ],
    dim=0,
)
# 收集 topk 索引张量
topk_index = torch.stack(
    [
        torch.as_tensor(
            req.output_topk_index[:num_states],
            device=batch.device,
            dtype=torch.int64,
        )
        for req in batch.reqs
    ],
    dim=0,
)

# 收集 hidden states，转移到 batch 设备
hidden_states = torch.stack(
    [req.hidden_states_tensor for req in batch.reqs], dim=0
).to(batch.device)

# 组装 EagleDraftInput
spec_info = EagleDraftInput(
    topk_p=topk_p,
    topk_index=topk_index,
    hidden_states=hidden_states,
    bonus_tokens=last_tokens_tensor,
)
spec_info.capture_hidden_mode = CaptureHiddenMode.LAST

# 若启用 overlap 调度，发布并暂存 future 信息
if batch.enable_overlap:
    spec_info.future_indices = batch.req_pool_indices
    future_map.publish(spec_info.future_indices, batch.seq_lens)
    future_map.stash(spec_info.future_indices, spec_info)

return spec_info
```

评论区精华

gemini-code-assist[bot] 提出了两条关于防御性检查的建议：

- 在 build_eagle_disagg_draft_input 中，若 batch.reqs 为空，torch.stack 会引发 RuntimeError（第 36 行附近）。

- 若 hidden_states_tensor 为 None，会引发 TypeError（第 51 行附近）。 截至合并时，这些评论未被回复或修复，潜在风险未解决。
- 空批次及 hidden states 缺失的防御性检查 (correctness): 无回复或修复，PR 已合并，风险未解决。

风险与影响

- 风险：
  1. 空批次崩溃风险：如果 batch.reqs 为空，torch.stack 会抛出 RuntimeError，但当前调度逻辑可能保证批处理不为空，缺少显式防御。
  2. hidden states 缺失风险：如果 req.hidden_states_tensor 为 None（如 capture 失败），torch.stack 会抛出 TypeError，且无 fallback 处理。
  3. 测试覆盖：本 PR 未添加单元测试，回归风险依赖现有集成测试套件。
  4. 兼容性：自定义算法需额外实现 build_disagg_draft_input 才能使用分离，现有自定义算法不受影响（默认返回 None 回退）。
- 影响：
  - 用户：之前无法在分离模式下使用自定义推测算法，现在可以正常使用。
  - 系统：代码结构更清晰，Eagle 专用逻辑与通用调度解耦，便于维护和扩展。
  - 团队：为后续添加新的推测算法（如 DFLASH、NGRAM 等）的分离支持提供了标准扩展点。
  - 风险标记：空批次崩溃风险，hidden states 缺失风险，缺少测试覆盖

关联脉络

- 暂无明显关联 PR