PR #33463 完整报告

sgl-project/sglang

Fix fractional simulated acceptance in DSpark

合并时间：2026-08-07 15:45

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33463

执行摘要

- 一句话：DSpark 分数模拟接受长度改为每步重采样
- 推荐动作：值得精读。这个 PR 虽然只有 21 行改动，但体现了两个可复用的设计决策：一是把私有的模拟接受长度采样函数公开为共享 API，避免 DSpark 与 MTP/DFlash 各自实现导致的口径漂移；二是 CUDA Graph 场景下“缓存 tensor + 就地 fill_ 刷新”的模式，兼顾分配开销与每次验证的动态性。若你负责投机解码或 benchmark 工具链，建议结合 SGLANG_SIMULATE_ACC_LEN 的文档理解其语义演变。

功能与动机

PR body 明确指出根因：DSpark 在分配缓存输出 buffer 时对 SGLANG_SIMULATE_ACC_LEN 只做一次 round，分数值因此变成固定整数，且缓存结果在后续 verify 步骤中从不刷新。这与 MTP、DFlash 使用的 match-expected 采样行为不一致，导致分数配置下模拟接受长度失真。

实现拆解

本 PR 的改动围绕三个文件展开：

1. 公开共享采样函数（python/sglang/srt/speculative/spec_utils.py）：将 _sample_simulated_acc_len 重命名为 sample_simulated_acc_len，作为 MTP、DFlash、DSpark 共用的公开 API；同步更新 generate_simulated_accept_index 内部的调用点。函数逻辑不变，仍支持 multinomial 与 match-expected 两种方法，并按 [1, max_len] 截断。
2. 更新 DFlash 调用方（python/sglang/srt/speculative/dflash_utils.py）：apply_dflash_simulated_acceptance 的导入与函数调用同步改为新公开名，行为不变。
3. 修复 DSpark 的模拟接受逻辑（python/sglang/srt/speculative/dspark_components/dspark_verify.py）：TargetVerifyExecutor._simulated_correct_len 不再把采样结果固化到缓存，而是每个 verify 步骤调用 sample_simulated_acc_len(self._simulate_acc_len, SIMULATE_ACC_METHOD, self.gamma + 1) 重新采样，并用 fill_ 就地写入缓存的输出 tensor；缓存重建条件从“容量或 dtype 变化”扩展为“容量、dtype 或 device 变化”，创建方式从 torch.full 改为 torch.empty 以配合 fill_。
4. 测试与配套：本次没有新增测试文件，回归保障依赖现有的 speculative-decoding 相关测试与预提交 CI（PR body 说明 full pre-commit suite 通过，NV pipelines 通过）。

关键文件：

- python/sglang/srt/speculative/dspark_components/dspark_verify.py（模块 DSpark 验证；类别 source；类型 core-logic；符号 _simulated_correct_len）：DSpark verify 路径的核心修复位置：模拟接受长度从缓存固化改为每步重采样，并补充 device 维度的缓存重建条件，是本 PR 的行为变更主体。
- python/sglang/srt/speculative/spec_utils.py（模块 投机采样；类别 source；类型 core-logic；符号 sample_simulated_acc_len）：将私有采样函数 _sample_simulated_acc_len 公开为 sample_simulated_acc_len，成为 MTP/DFlash/DSpark 共享的统一采样入口。
- python/sglang/srt/speculative/dflash_utils.py（模块 DFlash 验证；类别 source；类型 dependency-wiring；符号 apply_dflash_simulated_acceptance）：DFlash 调用方同步使用新公开函数名，保证改名后全仓一致。

关键符号：sample_simulated_acc_len, _simulated_correct_len

关键源码片段

python/sglang/srt/speculative/dspark_components/dspark_verify.py

DSpark verify 路径的核心修复位置：模拟接受长度从缓存固化改为每步重采样，并补充 device 维度的缓存重建条件，是本 PR 的行为变更主体。

```
def _simulated_correct_len(
    self, *, bs: int, dtype: torch.dtype, device: torch.device
) -> torch.Tensor:
    # 缓存输出 tensor，仅当容量、dtype 或 device 任一不匹配时才重建，
    # 避免每个 verify 步骤都重新分配 GPU 内存。
    buf = self._simulated_correct_drafts_buf
    if (
        buf is None
        or buf.numel() < bs
        or buf.dtype != dtype
        or buf.device != device
    ):
        buf = torch.empty((max(bs, 512),), dtype=dtype, device=device)
        self._simulated_correct_drafts_buf = buf

    # 每个 verify 步骤都重新采样 acceptance length：
    # match-expected 会按分数配置在 floor/ceil 间随机选取，
    # 保证长时间运行的平均接受长度贴近 SGLANG_SIMULATE_ACC_LEN。
    simulated_acc_len = sample_simulated_acc_len(
        self._simulate_acc_len, SIMULATE_ACC_METHOD, self.gamma + 1
    )
    # 就地 fill_ 复用缓存，返回同一底层存储的视图。
    return buf[:bs].fill_(simulated_acc_len - 1)
```

python/sglang/srt/speculative/spec_utils.py

将私有采样函数 _sample_simulated_acc_len 公开为 sample_simulated_acc_len，成为 MTP/DFlash/DSpark 共享的统一采样入口。

```
def sample_simulated_acc_len(
    simulate_acc_len: float,
    simulate_acc_method: str,
    max_len: int,
) -> int:
    """Sample a simulated acceptance length in [1, max_len]."""
    if simulate_acc_method == "multinomial":
        # 用正态分布围绕配置值采样，再 clamp 到 [1, max_len]。
        simulated_values = torch.normal(
            mean=simulate_acc_len,
            std=1.0,
            size=(1,),
            device="cpu",
        )
        simulated_values = torch.clamp(simulated_values, min=1.0, max=max_len)
        simulate_acc_len = int(simulated_values.round().item())
    elif simulate_acc_method == "match-expected":
        # 期望匹配配置值的平均数：按小数部分加权随机取 floor 或 ceil。
        # 例如配置 4.7 时，70% 概率取 5，30% 概率取 4。
        simulate_acc_len = max(1.0, min(max_len, simulate_acc_len))
        lower = int(simulate_acc_len // 1)
        upper = lower + 1 if lower < max_len else lower
        if lower == upper:
            simulate_acc_len = lower
        else:
            weight_upper = simulate_acc_len - lower
            weight_lower = 1.0 - weight_upper
            probs = torch.tensor([weight_lower, weight_upper], device="cpu")
            sampled_index = torch.multinomial(probs, num_samples=1)
            simulate_acc_len = lower if sampled_index == 0 else upper
    else:
        raise ValueError(f"Invalid simulate_acc_method: {simulate_acc_method}")
    return int(simulate_acc_len)
```

评论区精华

PR 的 review 评论为空，实质性技术讨论集中在 issue 评论中：

- 维护者 nvpohanh 在冲突产生后要求作者修复：could you fix the conflicts?，作者随后与 CI 机器人交互重跑失败任务。
- nvpohanh 在 NV pipelines 全部通过后请求其他维护者评审：All NV pipelines have passed. @hnyls2002 @kpham-sgl could you help to review this small change?，最终由 kpham-sgl 直接 approve，说明该修复被认为风险可控、改动清晰。
- 冲突修复与 CI 验收 (other): 作者解决冲突并重跑 CI，NV pipelines 通过后由 kpham-sgl 直接 approve。

风险与影响

- 风险：
  1. 缓存语义变化：_simulated_correct_len 现在每次调用都执行采样（含 torch.multinomial CPU 采样）并用 fill_ 写入，对 CUDA Graph 场景 fill_ 是图内可捕获的就地操作，但 CPU 采样发生在图外，每次 verify 多一次 CPU 开销（单次标量采样，成本极低）。
  2. benchmark 口径变化：SGLANG_SIMULATE_ACC_LEN 是 benchmark 专用开关，本次修复会让 DSpark 在分数配置下的行为从“固定整数”变为“逐次 match-expected 采样”，使用该开关对比历史数据的用户需要知道口径变化。
  3. 共享 API 改名：_sample_simulated_acc_len 从私有变为公开 sample_simulated_acc_len，仓库内调用点已同步，但外部或未同步的插件若引用旧私有名会失效；由于 spec_utils 属内部模块，风险较低。
  4. 测试缺口：没有新增针对 DSpark 分数模拟接受长度的单测，回归依赖现有测试与手工验证，后续重构可能回退此行为。
- 影响：影响范围集中在 speculative decoding 的 DSpark 路径与 benchmark 工具链：
- 用户 / 基准测试：使用 SGLANG_SIMULATE_ACC_LEN 分数配置的 DSpark 用户会获得更准确的模拟接受长度，如 PR 验证所示配置 4.7 时实际接受长度 4.68–4.78（此前会被固定为 4）。
- 系统与性能：不改变推理精度与吞吐，仅多一次 CPU 标量采样；TP8 CUDA Graph 场景验证 80/80 请求成功，吞吐 210.59 tokens/s。
- 团队：统一了 MTP/DFlash/DSpark 三类投机解码路径对模拟接受长度的采样口径，后续维护只需维护 sample_simulated_acc_len 单点。
- 风险标记：无新增测试覆盖，benchmark 路径行为变更，共享 API 改名

关联脉络

- PR #33650 [Kimi-K3] Allow DSPARK verify on cutedsl_mla (fold_sq): 同属 DSpark verify 路径的优化，且本 PR 的验证场景同样使用 Kimi K3，两者共同推动 DSpark 在 Kimi-K3 上的可用性。
- PR #33785 Fix Mistral-Large-3 EAGLE draft skipping DeepseekV2Model.init: 同属 speculative decoding 模块的修复，体现 python/sglang/srt/speculative/ 下的活跃维护与统一演进。