PR #30822 完整报告

sgl-project/sglang

[6/6][kimi-deterministic] Use deterministic seeded coins for EAGLE rejection sampling

合并时间：2026-07-24 17:11

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30822

执行摘要

- 一句话：使 EAGLE 验证阶段使用确定性种子硬币，实现推测解码可重现
- 推荐动作：本 PR 是 kimi-deterministic 系列的核心部分，值得精读。关键设计决策（哈希替代随机、浮点边界 clamp、列空间契约）展示了对采样内核确切行为的细致验证。建议阅读时关注 _seeded_verify_coins 的实现和测试文件中的结构化哈希验证方法，这对于构建确定性采样逻辑具有通用借鉴价值。同时注意配套的 MLP-sync 和 FA4 修正是确保整体确定性的必要前提。

功能与动机

EAGLE 验证阶段在拒绝采样中使用的硬币来自 torch.rand，因此即使为每个请求设置了 sampling_seed，推测输出仍无法跨运行重现。该 PR 是 kimi-deterministic 系列（共 6 个 PR）的第 6 部分，目标是通过确定性硬币使验证阶段 RNG 可种子化，从而在 top-k/greedy 草稿选择下实现完全确定性推断。

实现拆解

1. 添加 _seeded_verify_coins 函数（python/sglang/srt/speculative/eagle_utils.py）：使用 Triton 内核 murmur_hash32 基于种子、序列长度和列索引生成哈希，并转换为 [0,1) 区间内的浮点数。列 [0, draft_token_num) 用于拒绝采样，列 draft_token_num 用于最终采样。
2. 修改 _verify_coins 函数：判断是否设置 sampling_seed，若已设置则调用 _seeded_verify_coins，否则回退到 torch.rand。
3. 处理 float32 边界精度问题：将哈希值从 uint32 转换为 float32 时，前 129 个哈希值会向上舍入到 1.0，而采样内核期望半开区间 [0, 1)。通过 clamp(max=1.0-2**-24) 将最大值约束为 float32 中小于 1 的最大值。
4. 更新调用入口：在 eagle_sample 函数中，用新的 _verify_coins 替换直接调用 torch.rand_like 的逻辑。
5. 添加单元测试（test/registered/unit/spec/test_eagle_seeded_coins.py）：覆盖种子可重现性、不同种子 / 序列长度的发散性、列分割映射（结构化哈希验证）、未设置种子时仍使用 torch.rand，以及种子请求路由到 _seeded_verify_coins。
6. 配套调整：包括 MLP-sync pad/unpad 的修正（在 DP attention 下正确切片位置 / 序列长度张量）、TBO 子批处理清理新增的 _original_num_tokens 字段、FA4 注意后在

Blackwell 老架构上的 MLA 分派门控、确定性推理下禁用 fused_qkv_a_proj_with_mqa 内核、DeepGEMM BF16 路径支持确定性模式、以及解码 logprobs 改用 F.log_softmax 以实现与 prefill 的比特级别对齐。

关键文件：

- python/sglang/srt/speculative/eagle_utils.py（模块 推测解码；类别 source；类型 core-logic；符号 _seeded_verify_coins, _verify_coins, eagle_sample）：核心变更所在：新增 _seeded_verify_coins 和修改 _verify_coins，实现确定性硬币生成。
- test/registered/unit/spec/test_eagle_seeded_coins.py（模块 推测解码；类别 test；类型 test-coverage；符号 TestSeededVerifyCoins, test_seeded_coins_are_reproducible, test_distinct_seeds_or_positions_diverge, test_column_split_maps_rejection_then_final）：新增单元测试，锁定确定性硬币的各项契约：可重现性、种子 / 长度发散、列分割、未种子回退。
- python/sglang/srt/models/deepseek_common/attention_backend_handler.py（模块 注意力机制；类别 source；类型 data-contract；符号 handle_attention_fa4）：FA4 注意力后端确定性支持：在 Blackwell 架构上启用 MLA 分派，其他架构保留 MHA_chunked_KV 回退。
- python/sglang/srt/server_args.py（模块 配置管理；类别 source；类型 core-logic；符号 _handle_deterministic_inference）：确定性推理参数验证：强制 DeepSeek 模型在确定性模式下使用支持的注意力后端，并针对 FA4 检查架构。
- test/registered/unit/model_executor/test_mlp_sync_pad_unpad.py（模块 DP 注意力；类别 test；类型 test-coverage；符号 TestMlpSyncPadUnpad, test_decode_post_forward_unpads_per_request_tensors, test_extend_post_forward_unpads_positions）：新增针对 DP attention 下 MLP-sync pad/unpad 的单元测试，确保 post-forward 切片正确性，避免影响种子采样。

关键符号：_seeded_verify_coins, _verify_coins, eagle_sample, handle_attention_fa4, _handle_deterministic_inference

关键源码片段

python/sglang/srt/speculative/eagle_utils.py

核心变更所在：新增 _seeded_verify_coins 和修改 _verify_coins，实现确定性硬币生成。

```
# python/sglang/srt/speculative/eagle_utils.py (head excerpt)

def _seeded_verify_coins(
    *,
    sampling_seed: torch.Tensor,
    seq_lens: torch.Tensor,
    draft_token_num: int,
    device,
) -> Tuple[torch.Tensor, torch.Tensor]:
    """Derive deterministic verify-side coins from per-request sampling seeds.

    Mirrors the main seeded-sampling path: murmur_hash32(seed, seq_lens,
```

```
    column) mapped to [0, 1). Columns [0, draft_token_num) drive the
    per-draft rejection coins; column draft_token_num drives the final
    fallback-sampling coin.
    """
    from sglang.kernels.ops.sampling.murmur_hash import murmur_hash32

    cols = torch.arange(draft_token_num + 1, device=device, dtype=torch.int64)
    hashed = murmur_hash32(
        sampling_seed.to(torch.uint64), seq_lens.to(torch.uint64), cols
    )
    uniforms = hashed.to(torch.float64) / torch.iinfo(torch.uint32).max
    # The float32 cast rounds the top 129 uint32 hashes to exactly 1.0, but
    # the sampling kernels expect half-open [0, 1) coins: a 1.0 coin walks
    # past the last CDF bucket and can return a zero-probability token.
    # Clamp to the largest float32 below one; every other coin value is
    # untouched, so previously verified bitwise baselines stay intact.
    max_coin = 1.0 - 2**-24
    coins = (
        uniforms[:, :draft_token_num].to(torch.float32).clamp_(max=max_coin)
    ).contiguous()
    coins_for_final_sampling = (
        uniforms[:, draft_token_num].to(torch.float32).clamp_(max=max_coin)
    ).contiguous()
    return coins, coins_for_final_sampling


def _verify_coins(
    *,
    sampling_info: SamplingBatchInfo,
    seq_lens: torch.Tensor,
    draft_token_num: int,
    candidates: torch.Tensor,
    device,
) -> Tuple[torch.Tensor, torch.Tensor]:
    """Rejection and final-sampling coins for verify: deterministic seeded
    coins when sampling_seed is set, torch.rand otherwise.
    """
    if sampling_info.sampling_seed is not None:
        return _seeded_verify_coins(
            sampling_seed=sampling_info.sampling_seed,
            seq_lens=seq_lens,
            draft_token_num=draft_token_num,
            device=device,
        )
    # coins for rejection sampling
    coins = torch.rand_like(candidates, dtype=torch.float32, device=device)
    # coins for final sampling
    coins_for_final_sampling = torch.rand((bs,), dtype=torch.float32, device=device)
    return coins, coins_for_final_sampling
```

test/registered/unit/spec/test_eagle_seeded_coins.py

新增单元测试，锁定确定性硬币的各项契约：可重现性、种子 / 长度发散、列分割、未种子回退。

```
# test/registered/unit/spec/test_eagle_seeded_coins.py

class TestSeededVerifyCoins(CustomTestCase):
    def test_seeded_coins_are_reproducible(self):
        """相同种子和序列长度产生比特完全相同的硬币"""
        coins_a, final_a = _coins([12345, 67890, 12345], [7, 9, 7])
        coins_b, final_b = _coins([12345, 67890, 12345], [7, 9, 7])
        self.assertEqual(coins_a.shape, (3, DRAFT_TOKEN_NUM))
        self.assertEqual(final_a.shape, (3,))
        self.assertTrue(torch.equal(coins_a, coins_b))
        self.assertTrue(torch.equal(final_a, final_b))
        # 相同 (seed, seq_len) 对哈希相同，与行号无关
        self.assertTrue(torch.equal(coins_a[0], coins_a[2]))
        self.assertEqual(final_a[0].item(), final_a[2].item())
        # 硬币应在 [0, 1) 内
        self.assertTrue(bool((coins_a >= 0).all() and (coins_a < 1).all()))
        self.assertTrue(bool((final_a >= 0).all() and (final_a < 1).all()))

    def test_distinct_seeds_or_positions_diverge(self):
        """不同种子或不同长度产生不同的硬币"""
        coins, final = _coins([12345, 67890, 12345], [7, 9, 11])
        self.assertFalse(torch.equal(coins[0], coins[1])) # 不同种子
        self.assertFalse(torch.equal(coins[0], coins[2])) # 不同 seq_len

    def test_column_split_maps_rejection_then_final(self):
        """列分割契约：列 [0, draft_token_num) 为拒绝硬币，最后一列为最终采样硬币"""
        umax = torch.iinfo(torch.uint32).max

        def _structured_hash(seed, positions, col_indices):
            rows = torch.arange(seed.shape[0], device=seed.device).unsqueeze(1)
            return (rows * 1000 + col_indices.unsqueeze(0)).to(torch.uint32)

        with patch(
            "sglang.kernels.ops.sampling.murmur_hash.murmur_hash32",
            side_effect=_structured_hash,
        ):
            coins, final = _coins([1, 2], [3, 4])

        def _expected(row, col):
            return (
                torch.tensor(row * 1000 + col, dtype=torch.float64)
                .div(umax)
                .to(torch.float32)
                .item()
```

```
            )

        for row in range(2):
            for col in range(DRAFT_TOKEN_NUM):
                self.assertEqual(coins[row, col].item(), _expected(row, col))
            self.assertEqual(final[row].item(), _expected(row, DRAFT_TOKEN_NUM))

    def test_unseeded_requests_keep_torch_rand(self):
        """未设置种子时，硬币仍由 torch.rand 生成，两次调用不同"""
        with patch(
            "sglang.kernels.ops.sampling.murmur_hash.murmur_hash32"
        ) as mock_hash:
            coins_a, final_a = _verify_coins(...) # 未设置种子
            coins_b, final_b = _verify_coins(...)
        mock_hash.assert_not_called()
        self.assertFalse(torch.equal(coins_a, coins_b))
```

评论区精华

审查者 ch-wan 提出了三个核心评论：

- 草稿路径非确定性提醒：指出即使验证硬币确定，若使用 rejection sampling 草稿模式（fast_sample/torch.multinomial），不同运行仍会产生不同草稿，从而影响最终输出。作者回应当前更改仅覆盖验证侧 RNG，草稿侧随机性需后续 PR 处理。
- 建议增加单元测试：要求针对固定 sampling_seed 检查硬币可重现性、未设置种子仍用 torch.rand 以及列分割契约。作者添加了 test_eagle_seeded_coins.py，包含结构化哈希模拟来锁定列行为。
- float32 精度边界 bug：指出 hashed / UINT32_MAX 转换为 float32 时，最大 129 个 uint32 值会舍入为 1.0，而采样内核期望 [0, 1)，可能导致选择零概率 token。作者在后续提交中通过 clamp(max=1.0 - 2**-24) 修复此问题。
- 草稿路径非确定性提醒 (design)：作者确认当前更改仅覆盖验证侧 RNG，草稿侧需后续 PR 处理，PR body 已明确说明范围。
- 建议增加单元测试 (testing)：作者添加了 test/registered/unit/spec/test_eagle_seeded_coins.py，覆盖所有建议点。
- float32 精度边界 bug (correctness)：作者在后续提交中通过 clamp(max=1.0-2**-24) 修复，确保硬币严格在 [0, 1) 内。

风险与影响

- 风险：
  1. 浮点精度边界：原始转换存在硬币等于 1.0 的风险，已通过 clamp 修复，但需确保 clamp 值在各硬件上正确。
  2. 确定性依赖传播：EAGLE 验证侧确定性依赖于 sampling_seed 的正确传递，若上游采样信息丢失或覆盖可能导致意外回退到 torch.rand。

3. 草稿侧非确定性残留：当前仅覆盖验证 RNG，经典 rejection-sampling 草稿路径仍非确定，可能造成部分用户对“完全确定性”的误解。
4. 性能影响：仅影响设置种子的请求，哈希操作开销极低，但需注意大规模 batch 下的累积。
5. MLP-sync unpad 覆盖：新增的 _original_num_tokens 字段在 TBO 子批处理中需显式清理，已通过测试覆盖但可能遗漏其他构造点。 - 影响：用户视角：设置 sampling_seed 后，EAGLE 推测输出（top-k/greedy 草稿模式）变为可重现，便于调试、回放和 A/B 测试。rejection-sampling 草稿模式仍非确定，需后续改进。系统层面：变更集中在 eagle_utils.py，影响所有使用 EAGLE verify 的推理路径。DP attention 下的 MLP-sync 修正影响多卡通信 shape 对齐。FA4 和 DeepGEMM 的确定性门控影响 Blackwell 和其他架构上的确定性模型。团队协作：该 PR 集成了系列累积变更，代码评审中已解决主要设计权衡，后续需跟进草稿侧随机种子化。

- 风险标记：浮点精度边界，确定性分支覆盖，草稿侧非确定性残留，TBO 子批处理兼容性

关联脉络

- PR #30821 [5/6][kimi-deterministic] Compute decode logprobs via F.log_softmax: 本 PR 的前置依赖，同一系列的第 5 部分，实现解码 logprobs 的确定性对齐。
- PR #30820 [4/6][kimi-deterministic] MLP-sync unpad and TBO child batch fix: 同一系列的第 4 部分，修复 DP attention 下 MLP-sync 后处理问题。