PR #17260 完整报告

sgl-project/sglang

[Feature] [Ngram spec] Support ngram spec v2

合并时间：2026-06-10 17:46

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/17260

执行摘要

- 一句话：支持 ngram 猜测解码 v2，启用 overlap 调度
- 推荐动作：值得精读：此 PR 展示了如何将 ngram 猜测解码平滑融入现有的 spec v2 overlap 框架，设计决策值得学习，尤其是通过 Mixin 复用 EAGLE 的 pipeline、以及对非连续接受 token KV cache 的移动策略。关注点：move_accepted_tokens_to_target_kvcache 的语义、max_tree_depth 的动态推导。合并后建议：跟进 return_logprobs 支持，并补充 NPU/AMD 测试。

功能与动机

支持 ngram 猜测解码的 overlap 调度（v2），以缩短连续解码 batch 之间的 CPU 气泡，提升吞吐和延迟。相关讨论见 issue #11762 和 #21052。

实现拆解

实现分为以下步骤：

1. 继承 EAGLE v2 Mixin：NgramVerifyInput 改为继承 EagleDraftInputV2Mixin 和 EagleVerifyInputV2Mixin，从而复用 spec-v2 的 filter_batch、merge_batch、prepare_for_decode 等方法。移除原有的 prepare_for_verify、_fill_requests 等 v1 专用方法，大幅减少代码量。
2. 整合 NGRAMWorker：移除独立的 ngram_worker_v2，将 v2 逻辑（如 _prev_decode_rids 集合、enable_overlap 标志、move_accept_tokens_to_target_kvcache 调用）直接合并进 NGRAMWorker。__init__ 中新设 self.enable_overlap、self.req_to_token_pool 等属性。
3. 修改草稿准备与验证完成回调：_prepare_for_speculative_decoding 中根据是否 overlap 走不同分支；on_verify_complete_cpu 处理 logprob 计算和 KV cache 移动。move_accept_tokens_to_target_kvcache 用于将非连续接受的 token 的 KV cache 移动到正确 slot。
4. 适配通用验证采样：在 eagle_info_v2.py 的 sample() 中，根据 batch.spec_algorithm.is_ngram() 动态调整 max_tree_depth（ngram 无深度限制）和 topk（设为 -1 表示不启用 topk 约束）。
5. 配置与内存分配调整：arg_groups/speculative_hook.py 的 _handle_ngram 移除 server_args.disable_overlap_schedule = True，并自动根据 draft_token_num 和 topk

计算 speculative_num_steps。mem_cache/common.py 的 get_alloc_len_per_decode 添加 is_ngram 判断，使其在 page_size > 1 时也能使用平坦分配公式。

6. 测试与兼容性：保留 test_spec_ngram.py 和 test_spec_ngram_extra.py 的 v2 测试，并额外添加 no-overlap（同步 v2）变体。v1 路径完全移除，现有用户需升级配置（去掉手动 --disable-overlap-schedule 等）。

关键文件：

- python/sglang/srt/speculative/ngram_info.py（模块 猜测解码；类别 source；类型 dependency-wiring；符号 NgramVerifyInput, get_spec_adjust_token_coefficient, generate_attn_arg_prefill, filter_batch）：核心数据结构 NgramVerifyInput 重构：移除 v1 方法，继承 EAGLE v2 Mixin，添加 V2 属性 (future_indices, new_seq_lens, accept_tokens, accept_lens)，实现 filter_batch、merge_batch 等 v2 必备方法。
- python/sglang/srt/speculative/ngram_worker.py（模块 猜测解码；类别 source；类型 core-logic；符号 init, clear_cache_pool, _prepare_for_speculative_decoding, _prepare_draft_tokens）：工作线程核心改动：集成 spec v2 逻辑，包括 enable_overlap、内存池获取、_prev_decode_rids 集合、草稿准备 v2 分支、验证完成回调中调用 move_accept_tokens_to_target_kvcache 和 compute_spec_v2_logprobs。
- python/sglang/srt/speculative/eagle_info_v2.py（模块 猜测解码；类别 source；类型 core-logic；符号 sample）：采样方法适配 ngram：根据 is_ngram() 动态调整 max_tree_depth（允许链长到 draft_token_num）、贪婪模式下设置 topk=-1，模拟接受长度时修正宽度。
- python/sglang/srt/arg_groups/speculative_hook.py（模块 参数配置；类别 source；类型 configuration；符号 _handle_ngram）：移除强制禁用 overlap 调度的设置，并自动计算 speculative_num_steps 默认值。
- python/sglang/srt/speculative/spec_info.py（模块 猜测解码；类别 source；类型 core-logic；符号 supports_spec_v2, create_worker）：允许 ngram 返回 supports_spec_v2()=True，并移除 create_worker 中 overlap 时的 ValueError。
- python/sglang/srt/mem_cache/common.py（模块 内存缓存；类别 source；类型 core-logic；符号 get_alloc_len_per_decode）：调整 get_alloc_len_per_decode，使 ngram 不受 page_size>1 时的 topk>1 树分配逻辑影响，使用平坦分配。
- test/registered/spec/test_spec_ngram.py（模块 测试；类别 test；类型 test-coverage；符号 TestNgramSpeculativeDecodingPaged）：添加 v2 测试覆盖，调整配置以启用 overlap。
- test/registered/spec/test_spec_ngram_extra.py（模块 测试；类别 test；类型 test-coverage；符号 TestNgramSpeculativeDecodingNoOverlap）：添加 no-overlap（同步 v2）变体测试。
- python/sglang/srt/managers/overlap_utils.py（模块 调度器；类别 source；类型 core-logic；符号 _lazy_init_buf, set_draft_input_ngram）：添加 set_draft_input_ngram 方法用于设置 ngram 的 draft input 到缓冲区。
- python/sglang/srt/managers/schedule_batch.py（模块 调度器；类别 source；类型 core-logic）：新增 is_spec_v2 属性的访问条件放宽（spec_info 可空）。

- python/sglang/srt/model_executor/model_runner.py（模块 执行器；类别 source；类型 data-contract）：添加 tiny 适配使 cuda graph 支持 ngram v2（调整常量引用）。
- python/sglang/srt/model_executor/runner/decode_cuda_graph_runner.py（模块 执行器；类别 source；类型 data-contract）：类似适配，确保 cuda graph 捕获时分配正确。

关键符号：NgramVerifyInput.init, NgramVerifyInput.get_spec_adjust_token_coefficient, NgramVerifyInput.generate_attn_arg_prefill, NgramVerifyInput.filter_batch, NgramVerifyInput.merge_batch, NGRAMWorker.init, NGRAMWorker._prepare_for_speculative_decoding, NGRAMWorker._prepare_draft_tokens, NGRAMWorker.on_verify_complete_cpu, NGRAMWorker.forward_batch_generation, EagleVerifyInputV2Mixin.sample, _handle_ngram (in speculative_hook), get_alloc_len_per_decode (in common.py), supports_spec_v2 (in spec_info.py)

关键源码片段

python/sglang/srt/speculative/ngram_info.py

核心数据结构 NgramVerifyInput 重构：移除 v1 方法，继承 EAGLE v2 Mixin，添加 V2 属性 (future_indices, new_seq_lens, accept_tokens, accept_lens)，实现 filter_batch、merge_batch 等 v2 必备方法。

```
# file: python/sglang/srt/speculative/ngram_info.py

class NgramVerifyInput(SpecInput, EagleDraftInputV2Mixin, EagleVerifyInputV2Mixin):
    """Ngram 验证输入，通过 Mixin 复用 EAGLE v2 的 overlap 调度基础设施。"""

    def __init__(
        self,
        draft_token: torch.Tensor = None,
        custom_mask: torch.Tensor = None,
        positions: torch.Tensor = None,
        retrieve_index: torch.Tensor = None,
        retrieve_next_token: torch.Tensor = None,
        retrieve_next_sibling: torch.Tensor = None,
        draft_token_num: int = None,
        grammar: BaseGrammarObject = None,
        # V2 overlap 专属字段
        future_indices: Optional[torch.Tensor] = None,
        new_seq_lens: Optional[torch.Tensor] = None,
        accept_tokens: Optional[torch.Tensor] = None,
        accept_lens: Optional[torch.Tensor] = None,
    ):
        super().__init__(SpecInputType.NGRAM_VERIFY)
        self.draft_token = draft_token
        self.custom_mask = custom_mask
        self.positions = positions
        self.retrieve_index = retrieve_index
        self.retrieve_next_token = retrieve_next_token
```

```
        self.retrieve_next_sibling = retrieve_next_sibling
        self.draft_token_num = draft_token_num
        self.grammar = grammar

        # V2 overlap 数据：存储上一轮结果供下一轮草稿使用
        self.future_indices = future_indices
        self.new_seq_lens = new_seq_lens
        self.accept_tokens = accept_tokens
        self.accept_lens = accept_lens

        # 推导设备，兼容 custom_mask 为 None（v2 构造时可能仅有 new_seq_lens）
        self.device = (
            custom_mask.device if custom_mask is not None else new_seq_lens.device
        )

    def filter_batch(self, new_indices: torch.Tensor, has_been_filtered: bool = True):
        """按调度器过滤后的索引剪裁 spec_info 张量。"""
        if self.future_indices is not None:
            self.future_indices = self.future_indices[new_indices]
        if self.new_seq_lens is not None:
            self.new_seq_lens = self.new_seq_lens[new_indices]
        # accept_tokens 是扁平的一维张量，需要先 reshape 再剪裁
        self.accept_tokens = self.accept_tokens.reshape(-1, self.draft_token_num)[
            new_indices, :
        ]
        self.accept_tokens = self.accept_tokens.flatten()
        self.accept_lens = self.accept_lens[new_indices]

    def merge_batch(self, spec_info: NgramVerifyInput):
        """合并两个 spec_info（用于调度器合并 batch）。"""
        if self.future_indices is not None:
            assert spec_info.future_indices is not None
            self.future_indices = torch.cat(
                (self.future_indices, spec_info.future_indices), dim=0
            )
        if self.new_seq_lens is not None:
            assert spec_info.new_seq_lens is not None
            self.new_seq_lens = torch.cat(
                (self.new_seq_lens, spec_info.new_seq_lens), dim=0
            )
        self.accept_tokens = torch.cat(
            (self.accept_tokens, spec_info.accept_tokens), dim=0
        )
        self.accept_lens = torch.cat(
            (self.accept_lens, spec_info.accept_lens), dim=0
        )
```

python/sglang/srt/speculative/ngram_worker.py

工作线程核心改动：集成 spec v2 逻辑，包括 enable_overlap、内存池获取、_prev_decode_rids 集合、草稿准备 v2 分支、验证完成回调中调用 move_accept_tokens_to_target_kvcache 和 compute_spec_v2_logprobs。

```
# file: python/sglang/srt/speculative/ngram_worker.py

def _prepare_for_speculative_decoding(self, batch: ScheduleBatch):
    """准备 ngram 猜测解码的草稿和验证输入。"""
    ...
    # 构造 ngram 树掩码、位置等（原有逻辑）
    ...
    # 是否走 overlap v2 路径：需使用 spec-v2 风格的 NgramVerifyInput
    #  （带 accepted tokens 传递）
    if self.enable_overlap and batch.is_spec_v2:
        # 从上一轮 batch 中取出 accept_tokens / accept_lens
        # 存储到 spec_info 供下一轮草稿使用
        ...
        # 此处调用 move_accept_tokens_to_target_kvcache
        # 将非连续接受的 token 的 KV cache 移动到正确的 slot
        ...
    else:
        # 传统 v1 路径（已删除，目前全部走 v2）
        pass

def on_verify_complete_cpu(
    self, batch: ScheduleBatch, result: GenerationBatchResult
) -> GenerationBatchResult:
    """验证完成后的 CPU 回调：计算 logprob 并准备下一轮草稿所需的 accept 信息。"""
    verify_input: NgramVerifyInput = batch.spec_info
    ...
    # 将 accept_tokens 信息存入 spec_info，供 _prepare_for_speculative_decoding 下一轮使用
    accept_tokens = ...
    accept_lens = ...
    verify_input.accept_tokens = accept_tokens
    verify_input.accept_lens = accept_lens
    ...
    # 调用 compute_spec_v2_logprobs 代替此前的 add_output_logprobs_for_spec_v1
    ...
    return result
```

评论区精华

1. Overlap pipeline 设计：hnyls2002 质疑 ngram 没有 draft extend 阶段如何消除 CPU 气泡，SYChen123 解释 ngram v2 将 batching 与 draft 重叠，并通过分离 GPU 和 CPU 操作实现。
2. 准确性调试：SYChen123 发现 gsm8k 准确率下降，跟踪到 req.output_ids 在 overlap 调度下不完整，以及将 prev_tokens 插入 ngram trie 导致准确率骤降，最终定位根本问题是停止检查逻辑——v2 未正确触发 stop string，导致模型继续生成。

3. accept-length 问题：Ratish1 发现 _prepare_draft_tokens 中传给 NgramCorpus.batch_get 的上下文长度仅基于 origin_input_ids + output_ids，未包含 pending 的 prev_tokens，导致 trie 查询上下文不完整。
4. 性能测量：SYChen123 提供 nsys profile 显示 bubble gap 从 4.95ms（v1）降至 3.96ms（v2），并附 mock 压测结果（仅接受 1 token 时 TPOT 仍降低 4%）。
5. 代码清理：gemini-code-assist[bot] 多次建议移除中文注释、调试 print、注释掉的代码块，并修复类型提示问题。

- Overlap pipeline 设计（CPU 气泡消除）(design): 当前方案已可减少气泡，但完全消除仍需后续步骤；hnyls2002 建议先行合并当前改动，后续继续优化。
- 准确性调试：gsm8k 精度下降 (correctness): 通过不在 ngram trie 中插入 pending tokens 并正确实现停止检查，精度恢复正常。
- accept-length 上下文不足 (correctness): SYChen123 采纳建议，修正后 accept-length 恢复。
- 代码清理与类型安全 (style): SYChen123 在后续提交中逐步清理，但仍有少部分中文注释残留；最终版本已移除。
- logprobs 支持缺失 (testing): 标记为 TODO，当前版本不阻塞合并，后续修复。

风险与影响

- 风险：
  1. 回归风险：v1 路径被完全移除，所有 ngram 用户必须升级；若存在未覆盖的边界情况（如 page_size > 1 与 topk > 1 的组合），可能触发 CUDA illegal memory access（虽已修复）。
  2. 性能退化：overlap 调度下 CPU 开销仍然存在（batching 同步），但 profile 显示改善。极端负载下可能因 move_accepted_tokens_to_target_kvcache 引入额外 kernel 启动开销。
  3. Logprobs 支持不完整：PR 使用 compute_spec_v2_logprobs 代替旧的 add_output_logprobs_for_spec_v1，但开发者在讨论中指出 return_logprobs 仍未完全支持（标记为 TODO）。
  4. 适配性风险：NPU/AMD 平台未充分测试，assign_extend_cache_locs_func 可能返回 int32 而非预期的 int64 类型。 - 影响：对用户：使用 --speculative-algorithm NGRAM 的用户会自动启用 v2（overlap 调度），无需额外配置。如果之前手动指定了 --disable-overlap-schedule，需要移除该参数才能享受 v2 提升。建议进行回归验证。对系统：KV cache 分配逻辑变更，get_alloc_len_per_decode 现在对 ngram 使用平坦分配，影响内存预分配。eagle_info_v2.py 的 sample 方法为 ngram 动态调整 max_tree_depth 和 topk，对非 ngram 路径无影响。对团队：代码量大幅减少（由 +323/-502 变为净减少 179 行），维护性改善。测试覆盖了基本功能，但缺少对 return_logprobs 的测试。
- 风险标记：核心路径变更，数据契约调整，KV cache 移动新增 kernel，回归风险（v1 移除），logprobs 待支持

关联脉络

- PR #27764 [Spec] Extract move_accept_tokens_to_target_kvcache into spec_utils: 该 PR 提取了 move_accept_tokens_to_target_kvcache 函数，被本 PR 的 ngram v2 复用。
- PR #27695 Bundle set_kv_buffer write targets into KVWriteLoc (loc + swa_loc): 相关 KV cache 位置重构，本 PR 的 KV cache 移动逻辑依赖类似的底层抽象。
- PR #27617 [SWA] Cache full→SWA out_cache_loc per forward across attention backends: 同样涉及 speculative decoding 的 cache 位置优化，与本 PR 的 overlap 调度有协同。
- PR #25883 fix: forward update_mamba_state_after_mtp_verify in HybridAttnBackend: 之前修复的 spec v2 相关问题，本 PR 在 ngram 场景下复用了相同的 verify 完成处理模式。