PR #47093 完整报告

vllm-project/vllm

[Spec Decode] DSpark speculators checkpoint support

合并时间：2026-07-02 08:32

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47093

执行摘要

- 一句话：增加 DSpark speculators 格式 checkpoint 支持
- 推荐动作：值得精读。该 PR 展示了如何在现有投机解码框架中支持不同格式的检查点，涉及配置注册、模型接口设计、运行时适配三个层面。重点关注：compute_draft_logits/map_draft_to_target 的模型 - 运行时合约；reduced-vocab 时的散射采样策略；block 布局的动态选择。

功能与动机

支持从 HuggingFace 等源加载 speculators-format 的 DSpark 检查点（如 Qwen3-8B-speculator.dspark-reasoning），这些检查点使用与 dense DSpark 不同的 1+N fill-in block 布局，并可能具有 reduced draft vocabulary 及其到 target vocab 的映射表。需要适配 config、模型加载和采样逻辑以兼容这些检查点。

实现拆解

1. 注册 speculator 配置：在 vllm/transformers_utils/configs/speculators/algos.py 中新增 update_dspark 函数，通过 @register_speculator("dspark") 注册。该函数从 config_dict 提取 draft_vocab_size、markov_rank、block_size 等字段写入 pre_trained_config，并设置 dspark_bonus_anchor = True 以标记使用 1+N fill-in 布局。
2. 模型层适配：在 vllm/model_executor/models/qwen3_dspark.py 和 vllm/models/deepseek_v4/nvidia/dspark.py 中，修改 DSparkMarkovHead 使其接受独立的 draft_vocab_size 参数：markov_w1 嵌入 target vocab（完整大小），markov_w2 投影到 draft vocab。新增 compute_draft_logits 方法（返回 draft-vocab 的 logits，不经过 d2t 映射）和 map_draft_to_target 方法（将 draft id 映射到 target id）。权重加载时识别 d2t 张量并重命名为 draft_id_to_target_id，跳过 t2d（训练时用）。
3. 运行时适配：在 vllm/v1/worker/gpu/spec_decode/dspark/speculator.py 中，根据 dspark_bonus_anchor 标志选择 query 布局：False 时使用原来的 anchor-first（N slots），True 时使用 1+N fill-in（1 bonus + N slots）。新增 reduced-vocab 采样路径：在 load_draft_model 中预计算散射索引 _d2t_scatter_index（恒定的 draft→target 列偏移）和用 -inf 初始化的 _draft_scatter_buf；在 _sample_sequential 的循环中，若存在 _d2t_scatter_index，则将 draft logits 散射到 target vocab 中的对应列，然后采样得到 target id，直接用于 rejection sampling（省去一次映射）。

关键文件：

- vllm/model_executor/models/qwen3_dspark.py（模块 模型定义；类别 source；类型 data-contract；符号 DSparkMarkovHead.init, Qwen3DSparkModel.init, Qwen3DSparkForCausalLM.compute_draft_logits, Qwen3DSparkForCausalLM.map_draft_to_target）：DSpark 模型的 Qwen3 实现，核心变更包括 DSparkMarkovHead 支持独立 draft_vocab_size、新增 compute_draft_logits/map_draft_to_target 接口、权重加载添加 d2t 映射。
- vllm/transformers_utils/configs/speculators/algos.py（模块 配置；类别 source；类型 core-logic；符号 update_dspark）：注册 update_dspark 配置转换器，将 speculators-format 的配置字段映射为模型可用的配置，设置 dspark_bonus_anchor 标志。
- vllm/models/deepseek_v4/nvidia/dspark.py（模块 模型定义；类别 source；类型 data-contract；符号 DeepseekDSparkForCausalLM.compute_draft_logits, DeepseekDSparkForCausalLM.map_draft_to_target, DeepseekDSparkModel.init）：DeepSeek-V4 的 DSpark 模型适配，类似 Qwen3 的改动：支持 draft_vocab_size，新增 compute_draft_logits/map_draft_to_target（恒等映射）。
- vllm/v1/worker/gpu/spec_decode/dspark/speculator.py（模块 投机解码器；类别 source；类型 core-logic；符号 DSparkSpeculator.init, DSparkSpeculator.load_draft_model, DSparkSpeculator._sample_sequential）：DSpark speculator 运行时，核心改动包括动态选择 block 布局（anchor-first vs 1+N）、支持 reduced-vocab 概率采样（scatter + gumbel）。

关键符号：DSparkMarkovHead.init, Qwen3DSparkModel.init, Qwen3DSparkForCausalLM.compute_draft_logits, Qwen3DSparkForCausalLM.map_draft_to_target, Qwen3DSparkForCausalLM.load_weights, update_dspark, DeepseekDSparkForCausalLM.compute_draft_logits, DeepseekDSparkForCausalLM.map_draft_to_target, DSparkSpeculator.init, DSparkSpeculator.load_draft_model, DSparkSpeculator._sample_sequential

关键源码片段

vllm/model_executor/models/qwen3_dspark.py

DSpark 模型的 Qwen3 实现，核心变更包括 DSparkMarkovHead 支持独立 draft_vocab_size、新增 compute_draft_logits/map_draft_to_target 接口、权重加载添加 d2t 映射。

```
class Qwen3DSparkForCausalLM(DFlashQwen3ForCausalLM):
    # ... 其他代码不变

    def compute_draft_logits(self, hidden_states: torch.Tensor) -> torch.Tensor:
        # Draft-vocab logits without the d2t scatter:
        # the speculator adds the Markov bias in draft space,
        # then remaps via map_draft_to_target.
        return self.logits_processor(self.lm_head, hidden_states)

    def map_draft_to_target(self, draft_ids: torch.Tensor) -> torch.Tensor:
```

```
        # Map draft-vocab ids to target ids (identity for full-vocab drafts).
        if self.draft_id_to_target_id is None:
            return draft_ids
        # d2t mapping is stored as an offset table:
        # target_id = draft_id + draft_id_to_target_id[draft_id]
        return draft_ids + self.draft_id_to_target_id[draft_ids]

    def load_weights(self, weights: Iterable[tuple[str, torch.Tensor]]):
        model_weights = {}
        includes_embed_tokens = False
        includes_lm_head = False
        includes_draft_id_mapping = False
        for name, loaded_weight in weights:
            # t2d is training-only; the draft remaps via d2t at sampling time.
            if "t2d" in name:
                continue
            if "d2t" in name:
                # Rename d2t to draft_id_to_target_id
                name = name.replace("d2t", "draft_id_to_target_id")
                includes_draft_id_mapping = True
            elif "lm_head" not in name:
                name = "model." + name
            # ... 后续加载逻辑不变
```

vllm/transformers_utils/configs/speculators/algos.py

注册 update_dspark 配置转换器，将 speculators-format 的配置字段映射为模型可用的配置，设置 dspark_bonus_anchor 标志。

```
@register_speculator("dspark")
def update_dspark(config_dict: dict, pre_trained_config: dict) -> None:
    """
    将 speculators 格式的 DSpark 配置转换为 Transformers PreTrainedConfig 字典。
    设置必要的架构字段和参数，包括 draft 词汇表大小、markov head 参数、
    block 布局标志（dspark_bonus_anchor = True）等。
    """
    pre_trained_config["architectures"] = ["Qwen3DSparkModel"]
    # Speculators DSpark 使用 1+N fill-in 布局（anchor = bonus token）
    pre_trained_config["dspark_bonus_anchor"] = True

    aux_layer_ids = config_dict["aux_hidden_state_layer_ids"]
    pre_trained_config["eagle_aux_hidden_state_layer_ids"] = aux_layer_ids
    # DSpark 索引 target 层的方式是 aux_id - 1（与 dense config 一致）
    pre_trained_config["target_layer_ids"] = [i - 1 for i in aux_layer_ids]

    for key in (
        "draft_vocab_size",
        "target_hidden_size",
        "mask_token_id",
        "markov_rank",
```

```
        "markov_head_type",
        "block_size",
        "enable_confidence_head",
        "confidence_head_with_markov",
    ):
        if config_dict.get(key) is not None:
            pre_trained_config[key] = config_dict[key]
```

vllm/models/deepseek_v4/nvidia/dspark.py

DeepSeek-V4 的 DSpark 模型适配，类似 Qwen3 的改动：支持 draft_vocab_size，新增 compute_draft_logits/map_draft_to_target（恒等映射）。

```
class DeepseekDSparkForCausalLM(DSparkPretrainedModel):
    # ... 其他代码不变

    def compute_draft_logits(self, hidden_states: torch.Tensor) -> torch.Tensor:
        # Full-vocab draft: 直接使用 base logits，不需要 d2t 映射
        return self.compute_logits(hidden_states)

    def map_draft_to_target(self, draft_ids: torch.Tensor) -> torch.Tensor:
        return draft_ids # Full-vocab: draft ids 就是 target ids
```

评论区精华

- Markov head vocab 维度：@benchislett 追问为何 markov_w1 使用 vocab_size 而非 draft_vocab_size。@mgoin 解释 markov_w1 嵌入的是前一个 token（来自 target vocab），需要完整词汇表；markov_w2 投影到 draft vocab，所以使用 draft_vocab_size。
- compute_draft_logits 接口：@benchislett 质疑该方法的必要性，认为其它模型未类似实现。讨论后确认这是模型与 speculator 之间的合约，统一接口未来可以改进。
- Block 布局检测：@benchislett 指出通过 block_size 推断布局不够健壮。改为在 speculators config 翻译中直接设置 dspark_bonus_anchor 标志，运行时直接读取。
- Reduced-vocab scatter 数值安全性：@benchislett 询问 -inf 填充是否稳定。@mgoin 确认并在 load 时用 -inf 初始化缓冲一次，后续只 scatter 特定列，避免每步 fill 重复。
- Scatter 实现优化：@benchislett 建议用 torch.where 等原生算子融合。由于散射索引恒定，维持现有 scatter + 预初始化方案。
  - Markov head vocab 维度设计 (design): decided: markov_w1 使用 vocab_size（target vocab），markov_w2 使用 draft_vocab_size，由构造参数区分。
  - compute_draft_logits 接口必要性 (design): accepted: 作为合约方法保留，未来可考虑统一接口。
  - Block 布局动态检测的稳健性 (design): resolved: 添加 dspark_bonus_anchor 配置标志，运行时直接读取。
  - Reduced-vocab scatter 数值安全性 (correctness): resolved: 采用一次初始化缓冲 + 运行时仅 scatter 的策略。
  - Scatter 实现优化 (performance): resolved: 维持 scatter 方案，已通过预初始化优化。

风险与影响

- 风险：
  - 回归风险：修改了 DSparkMarkovHead 构造函数签名，原有 dense DSpark 检查点加载时若未提供 draft_vocab_size，需要 fallback 到 vocab_size。代码中已用 getattr(config, "draft_vocab_size", None) or config.vocab_size 处理，但需确认两个模型（Qwen3、DeepSeek-V4）的现有检查点兼容性。
  - 性能风险：reduced-vocab 采样路径引入了 scatter 操作和额外缓冲，但已在 load 时预分配，且散射索引恒定，运行时开销可控。
  - 数值风险：scatter 时使用 -inf 填充非目标列，需要验证 softmax 数值稳定性（预期无问题）。
  - 测试缺失：本次变更未包含直接对应的测试文件，reduced-vocab 分支的覆盖完全依赖手工验证。
- 影响：
  - 用户影响：用户现在可以使用 speculators-format 的 DSpark 检查点（如 Qwen3-8B-speculator），扩展了投机解码灵活性和模型选择。
  - 系统影响：投机解码模块新增配置分支和采样逻辑，但整体架构保持向后兼容，现有 dense DSpark 检查点不受影响。
  - 团队影响：新增代码约 140 行，与 DFlash 框架同构，维护成本有限。
  - 风险标记：核心路径变更，缺少测试覆盖，数值稳定性，兼容性回退

关联脉络

- PR #46995 [Spec Decode] DSpark: 本 PR (#47093) 在 #46995 的基础上增加了 speculators 格式 checkpoint 的支持，#46995 引入了 DSpark 投机解码的基础框架，本 PR 扩展现有 DSpark 以兼容 speculators 格式的检查点。