PR #35022 完整报告

vllm-project/vllm

[Core] Support structured outputs for beam search

合并时间：2026-06-12 21:56

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/35022

执行摘要

- 一句话：为 beam search 添加结构化输出（JSON schema）支持
- 推荐动作：值得精读。该 PR 展示了如何在已有生成算法（beam search）中集成 grammar 约束的典型模式，并讨论了性能权衡（状态重建 vs 状态克隆）。设计思路和评论中的性能优化点（缓存、冗余删除）对类似集成有参考价值。

功能与动机

用户需要在 beam search 中使用结构化输出（如 JSON schema）来约束生成结果，但 beam search 之前缺少对 grammar 约束的支持。Issue #34782 描述了此需求：现有 auto-regressive 生成无法满足，beam search 虽能提供更多候选却无法约束格式。此 PR 填补了该空白。

实现拆解

1. 扩展参数：在 vllm/sampling_params.py 的 BeamSearchParams 类中添加 structured_outputs: StructuredOutputsParams | None = None 字段，允许用户传入结构化输出配置。
2. 重构 beam_search 方法：在 vllm/entrypoints/generate/beam_search/offline.py 中，将原本的 inner loop 抽取为独立的 _beam_search_step 方法，以降低 beam_search 的复杂度，并为后续集成约束逻辑提供清晰的切入点。
3. 初始化结构化输出：新增 _init_beam_search_structured_output 方法，根据 StructuredOutputsParams 初始化后端（如 xgrammar），返回 backend、key 和全局 bitmask。在 beam_search 主流程中若发现 params.structured_outputs 不为 None，则调用此方法准备状态。
4. 每步约束采样：新增 _build_beam_sampling_params 方法，为每个 beam 基于当前已生成 token 重新构建 grammar 状态并计算 allowed_token_ids，从而生成受约束的 SamplingParams。新增 _bitmask_to_token_ids 辅助函数，将后端返回的 packed int32 bitmask 转换为 list[int]，以便注入 SamplingParams.allowed_token_ids。
5. 测试配套：在 tests/samplers/test_beam_search.py 中新增 test_beam_search_structured_output，使用 xgrammar 后端和一个简单 JSON schema 验证 beam search 输出严格符合 schema。

关键文件：

- vllm/entrypoints/generate/beam_search/offline.py（模块 beam 搜索；类别 source；类型 dependency-wiring；符号 _bitmask_to_token_ids, _beam_search_step, _init_beam_search_structured_output, _build_beam_sampling_params）：核心实现文件，重写了 beam_search 方法并新添多个辅助函数以集成结构化输出
- tests/samplers/test_beam_search.py（模块 beam 测试；类别 test；类型 test-coverage；符号 test_beam_search_structured_output）：新增测试用例验证结构化输出在 beam search 中的正确性
- vllm/sampling_params.py（模块 采样参数；类别 source；类型 core-logic）：扩展 BeamSearchParams 以支持结构化输出参数

关键符号：_bitmask_to_token_ids, _beam_search_step, _init_beam_search_structured_output, _build_beam_sampling_params

关键源码片段

vllm/entrypoints/generate/beam_search/offline.py

核心实现文件，重写了 beam_search 方法并新添多个辅助函数以集成结构化输出

```
# 引擎侧允许的最大 token ID 数量，与 v1 采样器中的 MAX_NUM_ALLOWED_TOKEN_IDS 保持一致
_MAX_NUM_ALLOWED_TOKEN_IDS = 1024

# 全局缓存，避免每次转换都重建 torch.arange 索引
_bitmask_cache: dict[int, tuple[torch.Tensor, torch.Tensor, torch.Tensor]] = {}

def _bitmask_to_token_ids(bitmask_row: torch.Tensor, vocab_size: int) -> list[int]:
    """将 packed int32 位掩码行转换为允许的 token ID 列表"""
    if vocab_size not in _bitmask_cache:
        indices = torch.arange(vocab_size)
        _bitmask_cache[vocab_size] = (
            indices,
            indices >> 5, # 计算字索引 (i // 32)
            indices & 31, # 计算位索引 (i % 32)
        )
    indices, word_indices, bit_indices = _bitmask_cache[vocab_size]
    mask = ((bitmask_row[word_indices] >> bit_indices) & 1).bool()
    return indices[mask].tolist()


class BeamSearchOfflineMixin(OfflineInferenceMixin):
    """Offline inference for beam search"""

    def beam_search(self, prompts, params, ...):
        # ... 前置参数提取 ...

        # 结构化输出相关状态
        structured_output_backend: StructuredOutputBackend | None = None
        structured_output_key = None
        structured_output_bitmask = None
```

```
# 如果用户传了结构化输出参数，则初始化后端并获取全局位掩码
if params.structured_outputs is not None:
    (
        structured_output_backend,
        structured_output_key,
        structured_output_bitmask,
    ) = self._init_beam_search_structured_output(
        params.structured_outputs, tokenizer
    )

# 每步复用的基础采样参数（不含 allowed_token_ids）
base_sampling_params = SamplingParams(
    logprobs=2 * beam_width,
    max_tokens=1,
    temperature=temperature,
    skip_clone=True, # 内部 beam search，安全跳过克隆
)
# ... 后续在 _beam_search_step 中根据每个 beam 的历史构建具体的采样参数
```

评论区精华

- 性能优化 - 缓存位掩码转换：AI Review 指出 _bitmask_to_token_ids 每次调用都执行 torch.arange(vocab_size) 是 O(V) 开销，作者添加了全局 _bitmask_cache 以 vocab_size 为 key 缓存索引和位运算结果。
- 冗余 grammar 编译：Review 指出 _init_beam_search_structured_output 中调用了 backend.compile_grammar 但返回对象未使用，且后续 _build_beam_sampling_params 会重新编译。作者移除了该冗余调用。
- 语法状态克隆不可用：AI Review 指出每个 step、每个 beam 都需重新 replay 所有 token 导致 O(N^2) 复杂度，建议支持状态克隆。作者确认后端不支持克隆，添加注释说明当前方案的局限性。
- logprobs 额外过滤的必要性：Review 怀疑手动 set 过滤是冗余的，因为 sampler 应该已经通过 allowed_token_ids 限制了 logits。作者通过分析 v1/sample/sampler.py 指出，raw_logprobs 在 allowed_token_ids_mask 作用之前已被捕获（第 82 行），因此必须手动过滤后才能送入 grammar 状态机。
- 提取 _beam_search_step 方法：DarkLight1337 建议将内部循环抽取为独立方法以提高可读性，作者采纳并提取了 _beam_search_step。
- 避免使用 verify()：DarkLight1337 质疑在 beam search 中调用 verify() 的合理性（它执行了多余验证）。作者替换为直接从 structured_outputs_config 获取 backend，简化了逻辑。
  - 性能优化 - 缓存位掩码转换 (performance): 作者添加了全局 _bitmask_cache 以 vocab_size 为 key 缓存索引和位操作，避免重复分配。
  - 冗余 grammar 编译 (performance): 作者移除了 _init_beam_search_structured_output 中的编译调用，因为每个 beam 的编译由 _build_beam_sampling_params 完成。
  - 语法状态克隆不可用 (design): 作者确认后端不支持克隆，添加了注释说明当前方案，未改变设计。

- logprobs 额外过滤的必要性 (correctness): 作者分析 v1/sample/sampler.py 发现 raw_logprobs 在 mask 应用前已捕获，必须手动过滤，保留了过滤逻辑。
- 提取 _beam_search_step 方法 (design): 作者提取了 _beam_search_step 方法。
- 避免使用 verify() (design): 作者替换为直接从 structured_outputs_config 获取 backend，避免调用 verify()。

风险与影响

- 风险：
  - 性能退化：每个 step 每个 beam 都需重新编译 grammar 状态并计算所有 token 的 bitmask，当 beam width 和 sequence length 较大时开销显著。当前后端不支持状态克隆，无法增量更新。
  - 缓存膨胀：_bitmask_cache 是全局字典，虽按 vocab_size 缓存索引，但若运行时出现多种 vocab_size 仍会累积，但通常数量有限，风险低。
  - 正确性边界：当 grammar 允许的 token 数超过引擎最大允许 ID 数（_MAX_NUM_ALLOWED_TOKEN_IDS=1024）时，会回退到 logprobs 过滤（首次 commit 已处理）——该回退路径未经充分测试。
  - 向后兼容性：新增字段 structured_outputs 默认为 None，对现有 beam search 用户无影响。
  - 影响：影响离线 API 中调用 LLM.beam_search() 的用户，现在可以传入 structured_outputs 约束生成格式（如 JSON schema）。对未使用结构化输出的现有用户无性能或行为影响。未来可扩展至 online serving 场景（已有 follow-up PR #36285）。系统架构无变化。
  - 风险标记：性能退化风险，状态重建开销，缺少性能测试

关联脉络

- 暂无明显关联 PR