

PR #38174 完整报告

vllm-project/vllm

[Feature] Universal speculative decoding for heterogeneous vocabularies (TLI)

合并时间: 2026-07-02 16:34

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/38174>

执行摘要

- 一句话: 实现异质词汇表间的通用投机解码 (TLI)
- 推荐动作: 建议重点关注 `vocab_mapping.py` 中的核心算法 (空间前缀探测、归一化、交集构建), 以及 `speculative.py` 中的配置验证逻辑。对于考虑使用异质草稿模型的团队, 此 PR 是关键里程碑。推荐精读源码和测试。

功能与动机

当前 vLLM 要求草稿模型与目标模型共享完全相同的词汇表, 严重限制草稿模型选择, 通常需从头训练专门草稿。TLI 算法 (ICML 2025 oral) 通过 token 级别交集实现异质词汇表间无损投机解码。例如 Qwen2.5 家族词汇重叠 99.8%, TLI 可达约 61% 草稿接受率。关联 issue #38173。

实现拆解

1. 配置层扩展

在 `vllm/config/speculative.py` 的 `SpeculativeConfig` 中添加 `use_heterogeneous_vocab: bool` 字段。`__post_init__` 中启用时, 为 draft 模型设置独立 tokenizer。`_verify_args` 校验: 仅当 `method='draft_model'` 且 `draft_sample_method='greedy'` 时可用。

2. 词汇映射模块

新增 `vllm/v1/spec_decode/vocab_mapping.py`, 定义 `VocabMapping`。通过 `_detect_space_prefix` 动态检测 BPE/ SentencePiece 空格前缀, `_normalize_token` 归一化。计算交集后生成三个张量: `draft_to_target_ids`、`target_to_draft_ids`、`intersection_mask_draft`。`constrain_draft_logits` 将非交集 logits 置为 `-inf`。

3. 集成 DraftModelProposer

在 `DraftModelProposer.__init__` (`vllm/v1/spec_decode/draft_model.py`) 中, 当标志开启时加载 tokenizer 并初始化 `VocabMapping`, 跳过词汇大小检查; 否则保持原逻辑。

4. 基类采样路径适配

在 `SpecDecodeBaseProposer` (`llm_base_proposer.py`) 中新增相关属性和分支。`_greedy_sample` 中应用约束和 ID 映射; `_sample_draft_tokens` 的概率路径同样约束但断言 `draft_probs` 为 `None`。

5. 测试与文档

新增 `test_vocab_mapping.py`, 测试 `_detect_space_prefix` 在真实 tokenizer 上的行为及异常回退。更新示例脚本和文档 (`README.md`、`draft_model.md`) 说明 TLI 用法。

关键文件:

- `vllm/v1/spec_decode/vocab_mapping.py` (模块 投机解码; 类别 source; 类型 core-logic ; 符号 `_detect_space_prefix`, `_normalize_token`, `_get_unk_token_id`, `VocabMapping`) : 核心新文件, 实现 TLI 算法的词汇映射逻辑, 包括空间前缀探测、归一化、交集构建和 logits 约束。
- `vllm/v1/spec_decode/draft_model.py` (模块 投机解码; 类别 source; 类型 data-contract ; 符号 `DraftModelProposer.init`) : 集成 TLI 的关键入口点, 在 `DraftModelProposer.__init__` 中根据标志条件初始化 `VocabMapping`。
- `vllm/v1/spec_decode/llm_base_proposer.py` (模块 投机解码; 类别 source; 类型 dependency-wiring; 符号 `SpecDecodeBaseProposer._greedy_sample`, `SpecDecodeBaseProposer._sample_draft_tokens`) : 在基类的 `_greedy_sample` 和 `_sample_draft_tokens` 中添加 TLI 约束和映射, 是推理路径的关键改动。
- `vllm/config/speculative.py` (模块 配置层; 类别 source; 类型 core-logic; 符号 `SpeculativeConfig.use_heterogeneous_vocab`, `SpeculativeConfig._verify_args`) : 添加 `use_heterogeneous_vocab` 配置字段和在 `_verify_args` 中增加关键验证逻辑。
- `tests/v1/spec_decode/test_vocab_mapping.py` (模块 测试; 类别 test; 类型 test-coverage; 符号 `test_detect_space_prefix_real_tokenizers`, `test_detect_space_prefix_fallback_on_failure`, `test_detect_space_prefix_empty_encode`) : 提供 `_detect_space_prefix` 的单元测试, 覆盖真实 tokenizer 和异常回退路径。
- `examples/features/speculative_decoding/spec_decode_offline.py` (模块 示例; 类别 source; 类型 configuration; 符号 `parse_args`) : 添加 `--use-heterogeneous-vocab` 命令行参数, 便于用户直接体验。
- `docs/features/speculative_decoding/README.md` (模块 文档; 类别 docs; 类型 documentation) : 在主 README 中添加 TLI 算法概要和使用说明。
- `docs/features/speculative_decoding/draft_model.md` (模块 文档; 类别 docs; 类型 documentation) : 在 `draft_model` 专用文档中添加 `heterogeneous vocab` 详细说明。

关键符号: `VocabMapping.init`, `_detect_space_prefix`, `_normalize_token`, `_get_unk_token_id`, `constrain_draft_logits`, `map_draft_to_target_ids`, `DraftModelProposer.init`, `SpecDecodeBaseProposer._greedy_sample`, `SpecDecodeBaseProposer._sample_draft_tokens`, `SpeculativeConfig._verify_args`

关键源码片段

`vllm/v1/spec_decode/vocab_mapping.py`

核心新文件, 实现 TLI 算法的词汇映射逻辑, 包括空间前缀探测、归一化、交集构建和 logits 约束。

```
# 探测 BPE/ SentencePiece 空格前缀
```

```

# BPE 使用 Ġ (U+0120), SentencePiece 使用 __ (U+2581)
def _detect_space_prefix(tokenizer) -> tuple[str, ...]:
    try:
        # 编码 " a" 来发现前缀字符
        ids = tokenizer.encode(" a", add_special_tokens=False)
        if ids:
            tok = tokenizer.convert_ids_to_tokens(ids[0])
            if isinstance(tok, str) and len(tok) > 1 and tok.endswith("a") and tok[0] not in (" ", "\t"):
                return (tok[:-1],)
    except Exception:
        pass
    # fallback: 覆盖 BPE 和 SentencePiece
    return ("\u0120", "\u2581")

def _normalize_token(token: str, prefixes: tuple[str, ...]) -> str:
    """去除空格前缀, 使得不同 tokenizer 中同一词的字面表示一致。"""
    for p in prefixes:
        if token.startswith(p):
            return " " + token[len(p):]
    return token

def _get_unk_token_id(tokenizer, role: str) -> int:
    """
    获取 UNK token ID, 优先 unk_token_id; 其次 eos_token_id; 最后抛 ValueError。
    使用 is not None 避免误判 token ID 0。
    """
    unk = getattr(tokenizer, "unk_token_id", None)
    if unk is not None:
        return unk
    eos = getattr(tokenizer, "eos_token_id", None)
    if eos is not None:
        logger.warning("VocabMapping: %s 无 unk_token_id, 回退到 eos_token_id=%d", role, eos)
        return eos
    raise ValueError(f"VocabMapping: {role} 既无 unk_token_id 也无 eos_token_id")

class VocabMapping:
    def __init__(self, target_tokenizer, draft_tokenizer, target_vocab_size, draft_vocab_size, device):
        self.target_unk = _get_unk_token_id(target_tokenizer, "target")
        self.draft_unk = _get_unk_token_id(draft_tokenizer, "draft")

        tgt_prefix = _detect_space_prefix(target_tokenizer)
        drf_prefix = _detect_space_prefix(draft_tokenizer)

        # 归一化并去重
        tgt_norm = {}
        for tok, tid in target_tokenizer.get_vocab().items():
            norm = _normalize_token(tok, tgt_prefix)
            if norm not in tgt_norm:

```

```

        tgt_norm[norm] = tid

    drf_norm = {}
    for tok, tid in draft_tokenizer.get_vocab().items():
        norm = _normalize_token(tok, drf_prefix)
        if norm not in drf_norm:
            drf_norm[norm] = tid

    # 计算归一化后的词汇交集
    common = set(tgt_norm.keys()) & set(drf_norm.keys())

    # 构建映射张量 (-1 表示无映射)
    drf2tgt = torch.full((draft_vocab_size,), -1, dtype=torch.long)
    tgt2drf = torch.full((target_vocab_size,), -1, dtype=torch.long)
    mask = torch.zeros(draft_vocab_size, dtype=torch.bool)

    for norm in common:
        t_id = tgt_norm[norm]
        d_id = drf_norm[norm]
        if t_id < target_vocab_size and d_id < draft_vocab_size:
            drf2tgt[d_id] = t_id
            tgt2drf[t_id] = d_id
            mask[d_id] = True

    self.draft_to_target = drf2tgt.to(device)
    self.target_to_draft = tgt2drf.to(device)
    self.intersection_mask = mask.to(device)
    self.intersection_size = int(mask.sum().item())

    def constrain_draft_logits(self, logits: torch.Tensor) -> torch.Tensor:
        """将非交集位置的 logits 置为 -inf, 保证 draft 只采样交集内 token."""
        logits[:, ~self.intersection_mask] = float('-inf')
        return logits

    def map_draft_to_target_ids(self, draft_ids: torch.Tensor) -> torch.Tensor:
        """将 draft token ID 映射到 target token ID, 未映射的替换为 target UNK."""
        mapped = self.draft_to_target[draft_ids]
        mapped[mapped == -1] = self.target_unk
        return mapped

```

vllm/v1/spec_decode/draft_model.py

集成 TLI 的关键入口点, 在 DraftModelProposer.__init__ 中根据标志条件初始化 VocabMapping。

```

from vllm.tokenizers.registry import get_tokenizer
from vllm.v1.spec_decode.vocab_mapping import VocabMapping

class DraftModelProposer(SpecDecodeBaseProposer):
    def __init__(self, vllm_config, device, runner=None):

```

```

super().__init__(vllm_config, device, pass_hidden_states_to_model=False, runner=runner)
self._raise_if_draft_tp_mismatch()

self.use_heterogeneous_vocab = self.speculative_config.use_heterogeneous_vocab

spec = self.speculative_config
if self.use_heterogeneous_vocab:
    # 异质词汇表：加载两个 tokenizer 并构建映射
    target_tokenizer = get_tokenizer(
        spec.target_model_config.tokenizer,
        trust_remote_code=spec.target_model_config.trust_remote_code,
    )
    draft_tokenizer = get_tokenizer(
        spec.draft_model_config.model,
        trust_remote_code=spec.draft_model_config.trust_remote_code,
    )
    self.vocab_mapping = VocabMapping(
        target_tokenizer=target_tokenizer,
        draft_tokenizer=draft_tokenizer,
        target_vocab_size=spec.target_model_config.get_vocab_size(),
        draft_vocab_size=spec.draft_model_config.get_vocab_size(),
        device=device,
    )
else:
    # 同质词汇表：继续原有的词汇大小一致性检查
    self._raise_if_vocab_size_mismatch()
    self.vocab_mapping = None

# ... 其余初始化保持不变

```

vllm/v1/spec_decode/llm_base_proposer.py

在基类的 `_greedy_sample` 和 `_sample_draft_tokens` 中添加 TLI 约束和映射，是推理路径的关键改动。

```

def _greedy_sample(self, hidden_states: torch.Tensor) -> torch.Tensor:
    if self.use_local_argmax_reduction:
        return self.model.get_top_tokens(hidden_states)

    if self.use_heterogeneous_vocab:
        logits = self.model.compute_logits(hidden_states)
        assert self.vocab_mapping is not None
        # 将非交集 logits 置为 -inf
        logits = self.vocab_mapping.constrain_draft_logits(logits)
        draft_token_ids = logits.argmax(dim=-1)
        # 将 draft ID 映射到 target ID 空间
        return self.vocab_mapping.map_draft_to_target_ids(draft_token_ids)

    return self.model.compute_logits(hidden_states).argmax(dim=-1)

```

```

def _sample_draft_tokens(self, hidden_states, sampling_metadata):
    if not self._enable_probabilistic_draft_probs or sampling_metadata.all_greedy:
        return self._greedy_sample(hidden_states), None

    logits = self.model.compute_logits(hidden_states)

    if self.use_heterogeneous_vocab:
        assert self.vocab_mapping is not None
        logits = self.vocab_mapping.constrain_draft_logits(logits)
        draft_token_ids, draft_probs = self._sample_from_logits(logits, sampling_metadata)
        draft_token_ids = self.vocab_mapping.map_draft_to_target_ids(draft_token_ids)
        # 配置已强制 greedy, 此断言为安全网
        assert draft_probs is None, "TLI 不支持概率 draft 采样"
        return draft_token_ids, draft_probs

    return self._sample_from_logits(logits, sampling_metadata)

```

评论区精华

@benchislett反对独立 Proposer 类型，要求使用标志集成减少维护。作者重构为配置标志方式。

@gemini-code-assist[bot]指出 `unk_token_id` 默认 0 对 Llama 错误，建议改抛异常。作者实现阶梯 fallback (`unk` → `eos` → `ValueError`)。

@jmamou(论文共同作者) 提出重复 token 去重、bonus token round-trip 等差距。去重已修复；bonus 映射计划后续 PR。

@benchislett要求禁止概率采样下使用 TLI 以防越界。配置层强制 `draft_sample_method='greedy'`，采样路径加断言。

@benchislett要求为 `_detect_space_prefix` 添加真实 tokenizer 测试。已添加参数化测试。

- 使用配置标志而非新 Proposer 类型 (design): 作者重构移除独立 proposer，改为在 `DraftModelProposer` 中条件分支。
- UNK token ID fallback 策略 (correctness): 实现 `unk` → `eos` → `ValueError` 阶梯逻辑。
- 概率采样场景的 TLI 安全性 (correctness): 配置验证强制 `draft_sample_method='greedy'`；采样路径加断言。
- 词汇映射的重复 token 去重 (correctness): 在 `VocabMapping` 中添加 if norm not in normalized 去重。
- 空间前缀检测需要测试覆盖 (testing): 添加 `test_vocab_mapping.py`，覆盖真实 tokenizer 和异常回退。
- Bonus token 文本级 round-trip 映射 (design): 未在当前 PR 实现，计划后续 PR。

风险与影响

- 风险:

- 词汇探测不确定性: `_detect_space_prefix` 依赖 `tokenize "a"`, 某些 `tokenizer` 可能返回意外结果导致 fallback 到双重前缀; 若实际前缀不在 fallback 列表则映射错误。测试覆盖主流模型家族。
- UNK 回退风险: 当 `unk_token_id` 缺失时回退到 `eos_token_id`, `eos` 可能不适宜作为未知 token 占位, 但仅影响 bonus token, 影响面小。
- 配置校验完整性: 若用户误配非 `greedy` 或非 `draft_model` 方法, 会得到明确错误提示, 风险低。
- 初始化性能开销: 需加载两个 `tokenizer` 并遍历完整词汇表 ($O(V)$), 大词汇表 (200k) 有数秒延迟, 但为一次性开销。
- 回归风险: 功能通过标志控制默认关闭, 不影响现有路径; 但修改了 `DraftModelProposer` 和基类热路径, 分支和断言有极小性能影响, 需维护双路径同步。
- 影响:
 - 用户影响: 用户可自由选择词汇重叠的小模型作为草稿, 无需训练或词汇对齐。仅需在 `speculative config` 设置 `use_heterogeneous_vocab=True`, 极大降低使用投机解码门槛。
 - 系统影响: 初始化解锁后占用少量 GPU 内存 (映射张量), 推理无额外开销。
 - 团队维护影响: 功能嵌入 `DraftModelProposer`, 后续对 `init` 或采样路径的修改需兼容 TLI 分支, 但通过标志和统一配置字段可控。
 - 风险标记: 核心路径变更, 配置验证依赖, `Greedy` 采样限制, UNK 回退语义风险, 词汇探测 fallback 不确定性

关联脉络

- PR #38173 [Feature]: Universal Speculative Decoding for Heterogeneous Vocabularies (TLI / Token-Level Intersection): 关联的 feature request issue, 提供动机和讨论基础。
- PR #46995 [Spec Decode] DSpark: 同为 v1 投机解码新方法, 涉及类似 `gpu_model_runner` 和 `proposer` 注册路径。