

PR #53896 完整报告

vllm-project/vllm

[Model] Support Qwen3.8-Flash-Next

合并时间: 2026-08-31 13:57

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/53896>

执行摘要

- 一句话: 新增 Qwen3.8-Flash-Next 支持: QSA 稀疏注意力、PLE 与混合 KV 缓存重构
- 推荐动作: 值得精读, 尤其是关注以下设计决策: `_get_packed_kv_cache_groups` 的打包分组语义、QSAIndexer 的 "权重无关" top-k 选择 + MTP skip_topk 跨步复用、PLE 的哈希 n-gram 词表布局、以及 `_get_ple_embedding_quant_method` 的量化分发门控 (它正是社区 NVFP4 混合量化 checkpoint 加载失败的根源)。对于想理解 vLLM 如何吸收 day-0 大模型并同步演进核心 KV 缓存抽象的人, 本 PR 是很好的案例; 但注意它同时叠加了大量未完全收敛的平台分支, 阅读时建议以 NVIDIA 路径为主线、以 AMD 路径为对照。

功能与动机

PR body 的目的是支持 HuggingFace 上的 Qwen/Qwen3.8-Flash-Next 模型, 提供 `vllm serve` 一键部署能力, 并配合 MTP 投机解码 (`num_speculative_tokens=3`) 与可选的 PLE CPU offload (`VLLM_PLE_CPU_OFFLOAD=1`)。该模型是首个同时包含 PLE 哈希 n-gram embedding、QSA 稀疏注意力、HyperConnection 残差结构与 MTP 的混合架构, 现有 vLLM 基础设施 (KV 缓存分组、Mamba 状态拷贝、模型 runner) 无法直接承载, 因此需要模型实现与核心缓存子系统同步改造。

实现拆解

实现可拆解为以下 5 个层次:

1. 模型注册与配置契约: 新增 `vllm/transformers_utils/configs/qwen4_exp.py` 与 `vllm/models/qwen4_exp/config.py`, 定义 `Qwen4ExpTextConfig`、`Qwen4ExpVisionConfig`, 并在 `_validate_ple_config`、`_validate_qsa_config`、`_validate_ple_layer_ids` 中做启动期校验 (如 PLE 层 id 合法性、QSA compress ratio 与 block size 整除关系), 保证配置错误在权重加载前暴露。
2. 三大新机制实现 (NVIDIA 与 AMD 各一套源码):
 - PLE 层 (`nvidia/ple_layer.py`、`amd/ple_layer.py`): 实现 `Qwen4ExpPLEGroupedNorm` 与 `Qwen4ExpNNGramEmbedding`, 后者用 `splitmix64` 哈希与素数词表布局 (`_splitmix64`、`_nth_prime_after`、`_make_vocab_layout`) 构造确定性 n-gram 查找表; NVIDIA 侧额外提供 FP8 embedding 方法 (`Qwen4ExpPLEFp8EmbeddingMethod`) 与量化分发门控 `_get_ple_embedding_quant_method`。

- QSA 稀疏注意力 (`common/qa_cache.py`、`nvidiaamd/qa.py`、`indexer_qa.py`) : `Qwen4ExpQSAFlashAttentionBackend` 继承 `FlashAttentionBackend` 但实际走自研 Triton kernel (`qa_sparse_paged_attention`) ; `QSAIndexer` 是权重无关的 token 选择器, 先投影 Q/K、压缩 key 存入 `QSACompressedKeyCache`, 再在压缩 key 上做 top-k 选择; `qa_cache.py` 提供 `circular_qa_slot_mapping`、`compressed_qa_slot_mapping` 与 Triton 元数据 kernel `_build_qa_metadata_kernel`。
 - HyperConnection (`common/hyperconnection.py`、`nvidiaamd/hyperconnection.py`) : `GatedResidual` 实现 mix / combine_and_mix / combine, 将前一层的 combine 与当前层的 RMSNorm 融合, 低秩 down+inject 投影合并为 `MergedColumnParallelLinear` 并物理 pad 到 16 行对齐。
3. MTP 草稿模型 (`nvidiaamd/mtp.py`) : `Qwen4ExpMultiTokenPredictor` 复用主模型 backbone, 但用 `residual_linear_shared` 融合 (`fc_embedding` + 共享 `fc_hidden`) 替代其他 MTP 的 `Linear(2H, H)`; 每个 draft step 输出两个 hidden 流 (`[T, H]` 单流给 `lm_head`、`[T, hc_count*H]` 多流给下一步) ; `set_skip_topk` 让 MTP 第 0 步做 QSA top-k 选择、后续步复用索引, `compact_topk_indices` 负责按请求对齐压缩索引行。
4. KV 缓存子系统重构 (核心基础设施) : `vllm/v1/core/kv_cache_utils.py` 重写 `_get_packed_kv_cache_groups`, 引入 `UniformTypeKVCacheSpecs` 打包分组, `MambaSpec.is_uniform_with_collection` 按 `page_size` 与 `tp_replicated` 拆分分组; `single_type_kv_cache_manager.py` 对 `CircularBufferSpec` 排除 zeroing 与 block 记录; KV Connector NIXL 侧修复 group spec 解包与 scratch cache 区域跟踪; 模型 runner V2 增加 `_prepare_ngram_context` 与 `_get_slot_mapping_mode` (对 `CircularBufferSpec` 返回 `SlotMappingMode.NONE`) , 最终合入前移除了 MRV1 集成, `Qwen4Exp` 仅支持 MRV2。
5. 量化、性能与测试配套: 支持 BF16 / FP8 / NVFP4 / MXFP4 权重格式, 新增 `nvidia/low_latency_gemm.py` (面向 SM103 的低延迟 GEMM 分发) , `fused_moe/oracle/unquantized.py` 中 `FlashInfer` TRTLLM 后端的 intermediate 维度 padding 从仅 act+silu 放宽到所有 MoE 配置; 同时新增 40+ 测试文件与 CI 配置 (含 `Qwen3.8-Flash-Next-FP8` 的 GPU CI lane) , 覆盖 PLE、QSA、MTP 与 KV 缓存打包。

关键文件:

- `vllm/models/qwen4_exp/nvidia/ple_layer.py` (模块 PLE 层; 类别 source; 类型 data-contract; 符号 `Qwen4ExpPLEGroupedNorm`, `Qwen4ExpPLEFp8EmbeddingMethod`, `create_weights`, `apply`) : PLE 层 NVIDIA 实现, 1256 行新增, 包含 FP8 embedding 方法、量化分发门控与 n-gram 前向; 社区关于 NVFP4/ 混合量化 checkpoint 加载失败的讨论全部集中在此。
- `vllm/models/qwen4_exp/amd/ple_layer.py` (模块 PLE 层; 类别 source; 类型 data-contract; 符号 `Qwen4ExpPLEGroupedNorm`, `Qwen4ExpNNGramEmbedding`, `_splitmix64`, `_is_prime_64`) : PLE 层 AMD 实现, 与 NVIDIA 版结构对称但缺少 FP8 embedding 方法; `zzw09773` 明确指出两个平台文件并不对称, 导致 FP8 checkpoint 在 ROCm 上无法加载 n-gram 表。
- `vllm/models/qwen4_exp/common/qa_cache.py` (模块 QSA 缓存; 类别 source; 类型 data-contract; 符号 `canonical_qa_rope_positions`, `_logical_positions`,

`_logical_to_physical_qsa_slots`, `circular_qsa_slot_mapping`) : QSA 侧缓存的核心：环形 raw key 缓存与压缩 key 缓存的 slot mapping、logical position 推导以及 Triton 元数据 kernel；社区报告的越界读与 `num_speculative_tokens=5` 启动失败都与这里的分组 / 整除逻辑有关。

- `vllm/models/qwen4_exp/nvidia/model.py` (模块 模型主体；类别 source；类型 data-contract；符号 `without_modelopt_fp4`, `_remap_qsa_cache_scale_name`, `Qwen4ExpSparseMoeBlock`, `Qwen4ExpDecoderLayer`) : 模型主体：按 `layer_type` 分发 `linear_attention/full_attention/MoE/MLP`，并接入 PLE、HyperConnection 与 QSA；`_remap_qsa_cache_scale_name` 处理 checkpoint scale 名到合并所有者的映射。
- `vllm/v1/core/kv_cache_utils.py` (模块 缓存分组；类别 source；类型 refactor；符号 `group_and_unify_kv_cache_specs`, `_get_kv_cache_groups_uniform_groups`, `_get_packed_kv_cache_groups`, `num_groups_for`) : 核心基础设施重构：重写 `_get_packed_kv_cache_groups` 引入 `UniformTypeKVCacheSpecs` 打包分组，LucasWilkinson 与 ivanium 的多轮 review 与连修 commit 都集中于此，影响所有混合模型。
- `vllm/models/qwen4_exp/nvidia/mtp.py` (模块 草稿模型；类别 source；类型 data-contract；符号 `_remap_ignored_layers`, `_remap_mtp_weight_name`, `_make_draft_vllm_config`, `Qwen4ExpMultiTokenPredictor`) : MTP 草稿模型：`residual_linear_shared` 融合、双流输出与 QSA `skip_topk` 跨步复用，是投机解码正确性的关键。
- `vllm/models/qwen4_exp/nvidia/qa.py` (模块 注意力后端；类别 source；类型 data-contract；符号 `Qwen4ExpQSA_MetadataBuilder`, `Qwen4ExpQSAFlashAttentionBackend`, `get_name`, `get_supported_kernel_block_sizes`) : QSA 注意力后端与实现：继承 `FlashAttentionBackend` 但实际走自研 Triton kernel，声明 `supports_kv_connector=False`；FP8 KV cache 未支持（社区 abraore 提出侧路补丁）。
- `vllm/models/qwen4_exp/amd/indexer_qsa.py` (模块 QSA 索引器；类别 source；类型 data-contract；符号 `apply_qsa_rope`, `apply_qsa_rmsnorm`, `QSAIndexer`, `output_width`) : AMD 侧 QSA indexer：用可移植的 `GemmaRMSNorm` 与 1D/MRoPE 组合替代 `FlashInfer` 依赖，forward 中先更新压缩 key 缓存再做 top-k 选择，是公式讨论中 `skip_topk` 复用的载体。

关键符号：`_get_ple_embedding_quant_method`, `Qwen4ExpNGramEmbedding.forward`, `QSAIndexer.forward`, `circular_qsa_slot_mapping`, `compressed_qsa_slot_mapping`, `_build_qsa_metadata_kernel`, `Qwen4ExpQSAFlashAttentionBackend.get_impl_cls`, `Qwen4ExpMultiTokenPredictor.forward`, `set_skip_topk`, `compact_topk_indices`, `_get_packed_kv_cache_groups`, `GatedResidual.combine_and_mix`, `_remap_qsa_cache_scale_name`, `_prepare_ngram_context`, `_get_slot_mapping_mode`

关键源码片段

[vllm/models/qwen4_exp/nvidia/ple_layer.py](#)

PLE 层 NVIDIA 实现, 1256 行新增, 包含 FP8 embedding 方法、量化分发门控与 n-gram 前向; 社区关于 NVFP4/ 混合量化 checkpoint 加载失败的讨论全部集中在此。

```
class Qwen4ExpPLEFp8EmbeddingMethod(QuantizeMethodBase):
    """FP8 PLE embedding with one global checkpoint scale."""

    def create_weights(
        self,
        layer: nn.Module,
        input_size_per_partition: int,
        output_partition_sizes: list[int],
        input_size: int,
        output_size: int,
        params_dtype: torch.dtype,
        **extra_weight_attrs,
    ) -> None:
        # PLE 的 n-gram 表在 checkpoint 里拆成 128 个 shard,
        # 运行时合并为单个 VocabParallelEmbedding, 因此这里注册
        # 一个 FP8 权重参数 + 一个 per-tensor 全局 scale 参数。
        weight_loader = extra_weight_attrs.get("weight_loader")
        weight = create_fp8_weight_parameter(
            sum(output_partition_sizes), input_size_per_partition, weight_loader
        )
        layer.register_parameter("weight", weight)
        weight_scale = create_fp8_scale_parameter(
            PerTensorScaleParameter,
            output_partition_sizes,
            input_size_per_partition,
            None,
            weight_loader,
            scale_dtype=torch.bfloat16,
        )
        layer.register_parameter("weight_scale", weight_scale)

    def apply(self, layer: nn.Module, x: torch.Tensor, bias=None) -> torch.Tensor:
        # PLE 表只参与 embedding lookup, 不参与 GEMM,
        # 所以线性算子路径直接拒绝。
        raise NotImplementedError("PLE FP8 weights only support embedding lookup")

    def embedding(self, layer: nn.Module, input_: torch.Tensor) -> torch.Tensor:
        # 读取时保持 FP8 原样取 row, 反量化交给后续算子。
        return F.embedding(input_, layer.weight)

    def _get_ple_embedding_quant_method(
        quant_config: QuantizationConfig | None,
        prefix: str,
    ) -> QuantizeMethodBase | None:
        """Select global-scale FP8 only for quantized PLE checkpoint shards."""
```

```

# 社区反馈 (OsakaTX / jschmied) : 混合量化 checkpoint (如 RadixArk
# NVFP4) 的 PLE 表以 FP8 shard 落盘, 但模型级 quant_config 是
# ModelOpt 而非 Fp8Config, 此门控会返回 None, 导致
# ngram_embedding.weight_scale 无处加载、启动失败。
if not isinstance(quant_config, Fp8Config):
    return None
if not quant_config.is_checkpoint_fp8_serialized:
    return None

ignored_layers = quant_config.ignored_layers
if is_layer_skipped(
    prefix,
    ignored_layers,
    quant_config.packed_modules_mapping,
    match_mode=quant_config.ignored_layers_match_mode,
):
    return None
# PLE checkpoint shards 在运行时合并为一个 embedding 参数。
shard_prefix = f"{prefix}.shard_"
if any(name.startswith(shard_prefix) for name in ignored_layers):
    return None
return Qwen4ExpPLEFp8EmbeddingMethod()

```

vllm/models/qwen4_exp/amd/indexer_qsa.py

AMD 侧 QSA indexer: 用可移植的 GemmaRMSNorm 与 1D/MRoPE 组合替代 FlashInfer 依赖, forward 中先更新压缩 key 缓存再做 top-k 选择, 是公式讨论中 skip_topk 复用的载体。

```

class QSAIndexer(nn.Module):
    """Replicated Q/K projection plus paged, weight-free QSA selection."""

    def __init__(self, *, vllm_config, config, layer_id, rotary_emb,
                 quant_config=None, prefix="") -> None:
        super().__init__()
        if vllm_config.cache_config is None:
            raise ValueError("QSA requires a paged KV cache")
        if vllm_config.model_config.dtype != torch.bfloat16:
            raise NotImplementedError("Qwen4Exp QSA currently requires BF16")
        self.token_topk = int(config.indexer_budget)
        self.compress_ratio = int(config.indexer_compress_ratio)
        # MTP 第 0 步选择目标行对齐的 token; 后续步复用这些索引,
        # 同时继续更新 QSA 侧缓存。
        self.skip_topk = False
        # 投影、norm、raw key cache 与 compressed key cache 的初始化 ...

    def forward(self, hidden_states, positions, out=None) -> torch.Tensor:
        """Return fixed-width request-relative token indices padded with -1."""
        metadata = self._metadata()
        if metadata is None:
            # MTP 后续步复用第 0 步 buffer; 普通路径返回全 -1 填充。

```

```

    if self.skip_topk and out is not None:
        return out
    result = torch.full(
        (hidden_states.shape[0], self.output_width),
        -1,
        dtype=torch.int32,
        device=hidden_states.device,
    )
    if out is not None:
        out.copy_(result)
        return out
    return result

raw_metadata, compressed_metadata = metadata
num_tokens = raw_metadata.num_actual_tokens
# 先投影 Q/K, 随后把 token K 写入 raw key 环形缓存、
# 按 compress_ratio 池化并归一化后写入 compressed key 缓存。
q, token_k = self.project_qk(
    hidden_states[:num_tokens], positions[... :num_tokens]
)
self._update_and_compress(
    token_k,
    positions[... :num_tokens],
    raw_metadata,
    compressed_metadata,
)
# skip_topk 时不再重跑 top-k kernel, 直接复用既有 buffer。
if self.skip_topk:
    if out is None:
        raise RuntimeError("QSA top-k reuse requires an output buffer")
    return out
return self._select(q, compressed_metadata, out)

```

评论区精华

合入前有多轮高质量 review, 最值得注意的几条:

- MRV2-only 路线: LucasWilkinson 在 APPROVE 时明确表态: "i do still think this should be an MRV2 only model due to the complexity of the MRV1 additions (ideally we want MRV1 to just be stable and in maintenance mode)". peakcrosser7 回应 "#53183 已让 MRV2 成为默认", 最终通过 commit [bdb03ff](#) 删除 MRV1 集成, 避免在维护模式的路线上继续叠加复杂度。
- CircularBufferSpec 是否排除 zeroing: vadiklyutiy 引用历史 (FullAttention→TQFullAttention→MLAAttention→HiddenStateCache 每次 "都觉得不会 NaN") 建议保留 zeroing; peakcrosser7 与 ivanium 则说明 QSA 环形槽只读本请求已写数据、且排除 zeroing 是为了规避 NIXL PD 下 block zeroing 与传输的竞态, 最终保留排除。

- 通用路径上的模型特定代码: LucasWilkinson 提醒 "we should try to avoid such model specific code here if possible; we spent alot of effort flushing out the DSv4 model specific code", 指向 kv_cache_utils 的 DeepSeek-V4 eagle 分组探测; ivanium 随后用 `_is_deepseek_v4_eagle` 收口到调用点。
- Kimi-K3 内核改动被拆出: vadiklyutiy 建议把 `vllm/v1/attention/backends/utils.py` 的 `_get_aligned_state_indices_kernel` 改动单独成 PR ("seems this is perf improvement only"), gau-nernst 直接 revert 了该文件与 Kimi-K3 相关测试, 保证本 PR 聚焦。
- FP8 MoE block-scale refine 的 hack: vadiklyutiy 指出 fp8.py 中 `weight_scale_refine` 逻辑 "a bit hacky", Isotr0py 承认 "This is just to make FP8 checkpoint work with day0, which isn't designed well" 并移出为独立 PR。
- FlashInfer padding 后是否仍值得选: vadiklyutiy 质疑 padding 后 FlashInfer TRTLLM 是否仍快于 Triton; peakcrosser7 给出实测表 (Prefill 4K -13.5%、Prefill 8K -26.7%、decode 最大 +3.5% 回退), 结论是 prefill 收益显著、decode 回退可接受, 保留 FlashInfer 为默认。
- PLE 权重加载绕过 weight_loader 契约: S1ro1 指出 `load_weights()` 直接 `copy_ple_embedding_shard_` 而非走 `weight_loader`, 会破坏 RL 权重更新生命周期; ivanium 以 commit `113ff01` 修复, 保证后续 RL 训练场景可正常更新 PLE 表。
 - CircularBufferSpec 是否应排除 block zeroing (design): 保留对 CircularBufferSpec 排除 zeroing 的改动, 理由 (自管理环形缓存 + 连接器竞态) 成立。
 - 是否应做成 MRV2-only 模型 (design): 采纳 MRV2-only: 合入前移除 MRV1 集成与对应测试。
 - Kimi-K3 元数据内核改动是否留在本 PR (design): 已 revert, 本 PR 不再触碰 Kimi-K3 元数据路径。
 - FP8 MoE block-scale refine 的 hack 是否可接受 (design): 从本 PR 移出, 放到独立分支 fp8-moe-block-scale-refine 单独提交。
 - FlashInfer TRTLLM padding 后是否仍优于 Triton (performance): 以实测数据为准, 保留 FlashInfer 默认; decode 回退被认为可接受。
 - PLE 权重加载绕过 weight_loader 契约 (correctness): 改为经由参数装载机写入, 保证未来 RL 训练可正确更新 PLE 表。
 - replicate_shared_expert 是否应为通用逻辑 (design): 逻辑保留在 qwen3_next, 作为受影响模型的局部处理; 未提升为通用机制。
 - PLE padded_buffer 切片是否可按 num_tokens 收紧 (performance): 采纳更窄的切片, 减少 forward 路径上的填充开销。

风险与影响

- 风险: 风险主要集中在三处:
 1. 异步调度下 ngram context 的正确性: zzw09773 在 8xMI325X 上实测默认配置 154 个 decode step 全部拿到错误上下文, 根因是 `_prepare_ngram_context` 在投机解码验收前读取 CPU 侧 token mirror, 拿到乐观的 -1 placeholder; 其给出的 `is_token_ids` sanitiser 又依赖 `enable_prompt_embeds` 开关且无法区分 placeholder 与真实 token。合入时该问

题的完整性修复状态需要确认，直接影响所有启用 MTP 的 Qwen3.8-Flash-Next 服务质量。

2. 核心路径重构回归面: `vllm/v1/core/kv_cache_utils.py` 的打包分组改动影响所有混合模型 (DeepSeek-V4、Kimi-K3、Jamba)，ivanium 随后连修 4 个问题 (NIXL group spec 解包、scratch cache 区域、block 截断断言、PP 下 mixer 权重)，说明重构初期确实引入了破坏；同 PR 内修复虽多，但长期回归风险仍需依赖 CI 覆盖。
3. 配置组合验证不完整：社区实测暴露多类未支持组合——NVFP4 + PLE CPU offload 不可用 (peakcrosser7 自认 "time constraints")、SM80 编译失败 (vowstar 报告的宏守卫不一致)、PP + MTP 非法内存访问、FP8 main KV cache 被显式 gate 拒绝、`num_speculative_tokens=5` 触发 QSA ring 整除断言失败。这些限制在 PR body 的验证表中没有完全如实呈现，容易误导用户选型。- 影响：影响范围很大：
 - 用户侧：可在 GB200/GB300/H200/MI355X 上以 BF16/FP8/NVFP4 运行 Qwen3.8-Flash-Next，并获得 MTP 投机解码与可选 PLE CPU offload；但 NVFP4 + offload、SM80、PP + MTP 等组合会启动失败或崩溃，需要严格对照验证矩阵选型。
 - 系统侧：KV 缓存打包分组机制新增 `UniformTypeKVCacheSpecs` 抽象与 `CircularBufferSpec` 规格，影响所有混合模型的缓存规划；模型 runner V2 新增 PLE 上文准备与 slot mapping 模式判断，`_record_new_block_ids` 的行为因 `CircularBufferSpec` 而改变。
 - 团队侧：这是 vLLM 首次以 Qwen4Exp 内部代号合入的阿里 + AMD + Inferact 三方协作模型，后续必然有一批 follow-up PR (PLE offload 完善、FP8 KV cache、SM80 支持、FP8 block-scale refine 独立提交等)，维护者需要持续跟进。
 - 风险标记：核心路径变更，异步调度正确性风险，量化组合支持不完整，跨平台实现不对称，预构建镜像漂移，配置验证矩阵不透明，模型特定代码外溢

关联脉络

- PR #53899 [Feat] PLE CPU offload: PR body 明确说明待合入，`VLLM_PLE_CPU_OFFLOAD=1` 依赖它的 PLE offload worker 与连接器；社区多个实机反馈都基于这两个 PR 的组合镜像。
- PR #53982 Kernel-side guard for compute slot mapping: zzw09773 在评论中单独 filing 的 issue: `ComputeSlotMappingKernel` 对 block 表窄于序列的越界读防护，是本 PR `CircularBufferSpec` 模型侧修复的防御纵深。
- PR #53183 Make MRV2 the default model runner: LucasWilkinson 与 peakcrosser7 讨论 MRV2-only 路线时引用，是删除本 PR MRV1 集成的直接依据。
- PR #50175 [1/N][warmup][DSv4] Migrate generic MLA metadata and indexing kernels: 与本次 KV 缓存 / runner / indexer 基础设施同一条演进线，LucasWilkinson 提到的 "清除 DSv4 模型特定代码" 工作即来自该系列。
- PR #54646 [Core][MRV2] Freeze gc during V2 CG capture; skip per-descriptor cleanup: MRV2 性能与 CUDA graph 捕获路径的配套演进，Qwen4Exp 的 QSA 元数据与 CG 捕获同样依赖该路径。
- PR #53155 [ROCm] Keep GLM-5.2 on MRV1 and disable default breakable cudagraph: MRV1/MRV2 双轨维护策略的对照案例：本 PR 把 Qwen4Exp 限制在 MRV2，与 GLM-5.2 留在 MRV1 形成同一决策框架下的两面。