PR #6453 完整报告

verl-project/verl

[veomni] feat: add VeOmni-native critic support

合并时间：2026-05-24 14:14

原文链接：http://prhub.com.cn/verl-project/verl/pull/6453

执行摘要

- 一句话：新增 VeOmni 原生 critic 训练支持
- 推荐动作：本 PR 是 VeOmni 功能生态的重要扩展，具有中等重要度。建议：
  - 核心开发人员精读 VeOmniEngine 的共享方法提取和类属性设计，该模式可推广到其他引擎。
  - 关注后续是否补充单元测试或集成测试，特别是 VeOmniEngineWithValueHead 的各种配置组合。
  - 考虑在文档中说明 VeOmniCriticConfig 的使用方法和参数要求。

功能与动机

VeOmniCriticConfig 在 veomni_critic.yaml 中被引用但从未实现，本 PR 填补了这一缺口，使得 VeOmni 引擎能够原生支持批评者 /value 模型训练。同时修复了 FSDP 引擎中基于类名字符串比较的条件分支可能导致的 position_ids 双重切片问题。

实现拆解

1. 配置层：在 verl/workers/config/critic.py 中新增 VeOmniCriticConfig 数据类，继承自 CriticConfig，默认策略设为 "veomni"，并通过 __post_init__ 将引擎配置指向 VeOmniEngineConfig。同时实现了 validate 方法，在非动态 batch 场景下校验 sequence parallelism size 与 GPU 数量的兼容性。
2. 引擎注册：在 verl/workers/engine/veomni/transformer_impl.py 中新增 VeOmniEngineWithValueHead 类，通过 @EngineRegistry.register(model_type="value_model", backend=["veomni"], device=["cuda", "npu"]) 注册。该类同时继承 VeOmniEngine（提供 VeOmni 通用逻辑）和 FSDPEngineWithValueHead（提供价值头逻辑）。
3. 逻辑抽取：在基类 VeOmniEngine 上添加类属性 _veomni_handles_position_ids = True，并提取 _apply_veomni_input_transforms 方法，封装了 VLM 掩码处理、序列并行 sharding 和 Flash Attention kwargs 共用逻辑。VeOmniEngineWithLMHead 和 VeOmniEngineWithValueHead 各自的 prepare_model_inputs 都调用此方法，消除重复。
4. FSDP 修复：在 verl/workers/engine/fsdp/transformer_impl.py 的 FSDPEngineWithLMHead.prepare_model_inputs 中，将 skip_position_ids_rmpad 的赋值从脆弱的 self.__class__.__name__== "VeOmniEngineWithLMHead" 改为 getattr(self, "

_veomni_handles_position_ids", False)，确保所有 VeOmni 子类正确跳过 SP 侧的 position_ids 切片。

5. 导出更新：在 verl/workers/engine/veomni/__init__.py 中新增 VeOmniEngineWithValueHead 的导入和导出，保证模块可见性。

关键文件：

- verl/workers/engine/veomni/transformer_impl.py（模块 引擎；类别 source；类型 core-logic；符号 _apply_veomni_input_transforms, VeOmniEngineWithValueHead, prepare_model_inputs）：核心引擎实现：新增 VeOmniEngineWithValueHead 类，提取共享方法 _apply_veomni_input_transforms，修复类名检查。
- verl/workers/config/critic.py（模块 配置；类别 source；类型 dependency-wiring；符号 VeOmniCriticConfig, post_init, validate）：配置类：新增 VeOmniCriticConfig，包含策略、引擎配置和验证逻辑。
- verl/workers/engine/veomni/__init__.py（模块 入口；类别 source；类型 dependency-wiring）：导出新引擎类，保证模块可访问。
- verl/workers/engine/fsdp/transformer_impl.py（模块 引擎；类别 source；类型 core-logic）：修复类名检查为属性检查，消除 position_ids 双重切片风险。

关键符号：VeOmniEngine.init, VeOmniEngine._apply_veomni_input_transforms, VeOmniEngineWithValueHead.prepare_model_inputs, VeOmniCriticConfig.post_init, VeOmniCriticConfig.validate, FSDPEngineWithLMHead.prepare_model_inputs (modified)

关键源码片段

verl/workers/engine/veomni/transformer_impl.py

核心引擎实现：新增 VeOmniEngineWithValueHead 类，提取共享方法 _apply_veomni_input_transforms，修复类名检查。

```
# verl/workers/engine/veomni/transformer_impl.py

class VeOmniEngine(FSDPEngine):
    # 类属性标志：VeOmni 自己处理 position_ids 的 SP 切片
    # 替代之前脆弱的 self.__class__.__name__ 比较
    _veomni_handles_position_ids = True

    def _apply_veomni_input_transforms(self, model_inputs: dict, micro_batch: TensorDict):
        """共享给 LM 和 Value 头的 VeOmni 输入变换逻辑。
           处理 VLM 掩码、序列并行分片、Flash Attention kwargs。
        """
        input_ids_rmpad = model_inputs["input_ids"]
        sp_enabled = parallel_state.get_parallel_state().sp_enabled
        sp_shard_collator = OmniSequenceShardCollator() if sp_enabled else None

        # 如果是 VLM 模型，添加 image/video 掩码并分片
        if self.module.config.model_type in VL_TYPE2INDEX.keys():
            image_mask = input_ids_rmpad == VL_TYPE2INDEX[self.module.config.model_type]
```

```
                ["IMAGE_INPUT_INDEX"]
                video_mask = input_ids_rmpad == VL_TYPE2INDEX[self.module.config.model_type]
                ["VIDEO_INPUT_INDEX"]
                model_inputs.update({"image_mask": image_mask, "video_mask": video_mask})
                if sp_enabled:
                    sp_shard_collator(model_inputs)

            # 处理 remove padding 时的 Flash Attention 参数
            use_remove_padding = tu.get_non_tensor_data(data=micro_batch, key="use_remove_
            padding", default=True)
            if use_remove_padding and model_inputs.get("position_ids", None) is not None:
                model_inputs.update(_prepare_veomni_flash_attention_kwargs(model_inputs["position_
                ids"]))
                if sp_enabled:
                    model_inputs["position_ids"] = sp_shard_collator.sp_slice(model_inputs["position_ids"]
                    , dim=-1)

    @EngineRegistry.register(model_type="value_model", backend=["veomni"], device=["cuda", "npu"])
    class VeOmniEngineWithValueHead(VeOmniEngine, FSDPEngineWithValueHead):
        """VeOmni 价值模型引擎，使用 FSDP2 + 序列并行。"""

        def prepare_model_inputs(self, micro_batch: TensorDict):
            # 调用父类（FSDPEngineWithValueHead）的 prepare_model_inputs 得到基础输入
            model_inputs, output_args = super().prepare_model_inputs(micro_batch)
            # 应用 VeOmni 共享输入变换（VLM 掩码、SP 分片、FA kwargs）
            self._apply_veomni_input_transforms(model_inputs, micro_batch)
            # 价值模型不需要 return_log_probs 和 fused_kernels，此处略去
            return model_inputs, output_args
```

verl/workers/config/critic.py

配置类：新增 VeOmniCriticConfig，包含策略、引擎配置和验证逻辑。

```
    # verl/workers/config/critic.py

    @dataclass
    class VeOmniCriticConfig(CriticConfig):
        """VeOmni 批评者模型训练配置，继承自 CriticConfig。
           使用 VeOmni 的 FSDP2 + 序列并行引擎。
        """
        strategy: str = "veomni"
        veomni: VeOmniEngineConfig = field(default_factory=VeOmniEngineConfig)
        grad_clip: float = 1.0

        def __post_init__(self):
            super().__post_init__()
            # 将 engine 指向 veomni 配置，供上层统一使用
            self.engine = self.veomni

        def validate(self, n_gpus: int, train_batch_size: int):
```

```
        """验证 VeOmni 批评者配置的合法性。"""
        super().validate(n_gpus, train_batch_size)
        if not self.use_dynamic_bsz:
            sp_size = self.veomni.ulysses_parallel_size
            if self.ppo_micro_batch_size is not None:
                if self.ppo_micro_batch_size * sp_size < n_gpus:
                    raise ValueError(
                        f"critic.ppo_micro_batch_size ({self.ppo_micro_batch_size}) * "
                        f"veomni.ulysses_parallel_size ({sp_size}) must be >= n_gpus ({n_gpus})"
                    )
```

评论区精华

Review 中的两条核心建议均已解决：

1. VeOmniCriticConfig 缺少 validate 方法：gemini-code-assist[bot] 指出没有配置验证可能导致运行时错误，作者 Luosuu 在 force-push 中添加了 validate 方法（检查 SP 与 GPU 数量的兼容性）。
2. VeOmniEngineWithValueHead 的输入准备逻辑重复：gemini-code-assist[bot] 建议提取共享逻辑，作者将 VLM/SP/FA 处理统一到 _apply_veomni_input_transforms 方法中，两个引擎子类均调用，消除了维护风险。

- VeOmniCriticConfig 缺失 validate 方法 (correctness): Luosuu 在 force-push 中添加了 validate 方法，包含 SP 与 GPU 数量的校验。
- VeOmniEngineWithValueHead 重复输入准备逻辑 (design): Luosuu 提取了 _apply_veomni_input_transforms 到基类 VeOmniEngine，两个子类共享。

风险与影响

- 风险：主要风险在于：
  1. 新引擎回归：VeOmniEngineWithValueHead 是新注册的引擎，缺乏大规模训练测试，可能在极端配置或模型下暴露问题。
  2. 配置验证覆盖面：VeOmniCriticConfig.validate 仅在非动态 batch 时检查，若用户启用 use_dynamic_bsz 则不生效，可能仍存在非法配置。
  3. position_ids 处理：将类名检查改为属性检查后，若其他第三方子类未设置 _veomni_handles_position_ids 但实际期望跳过切片，可能引入 bug。但此类子类极少，影响有限。
  4. 无配套测试：本次变更没有直接对应的测试文件，需依赖用户端验证。 - 影响：影响范围限于使用 VeOmni 引擎进行批评者（价值模型）训练的用户。对现有 FSDP 和 Megatron 训练流程无影响。FSDPEngineWithLMHead.prepare_model_inputs 的修改虽然影响所有 FSDP 子类，但行为保持向后兼容（默认 False），仅 VeOmni 变体受益。团队成员需要更新配置以使用新的 VeOmniCriticConfig。 - 风险标记：新引擎回归风险，配置验证覆盖不全，position_ids 属性依赖

关联脉络

- PR #6323 [veomni] feat: add veomni qwen3-30b and fix ep: 此前 VeOmni 引擎的基础支持，本次 PR 在此基础上增加了批评者支持。