PR #6051 完整报告

verl-project/verl

[trainer,cfg,rollout,algo] feat: Multi-Teacher OPD

合并时间：2026-04-20 12:31

原文链接：http://prhub.com.cn/verl-project/verl/pull/6051

执行摘要

- 一句话：实现多教师在线策略蒸馏框架
- 推荐动作：建议所有涉及蒸馏功能或资源编排的工程师精读此 PR。核心设计模式（资源池分割、请求路由、配置数据契约）值得借鉴。特别关注 _validate_replica_node_alignment 和 _resolve_teacher_key 的实现，它们体现了对多节点部署的严谨考虑。在部署多教师场景前，建议增加集成测试覆盖节点对齐和路由降级路径。

功能与动机

PR body 指出需要 'Classes and configs for managing multiple sets of teacher models and servers for multi-teacher OPD'，动机是让一个学生能从多个教师学习不同能力（如纯文本数学推理与多模态几何推理），每个样本可基于数据源字段路由到最合适的教师。评论中作者也强调教师服务器流式计算的隔离优势。

实现拆解

1. 配置数据契约重构：在 verl/workers/config/distillation.py 中将 DistillationTeacherModelConfig 增加 key（路由标识）、num_replicas（副本数）、per_replica_world_size（计算属性）等字段；DistillationConfig 新增 teacher_models 字典（按 key 索引的教师配置）、teacher_key（样本路由字段名）和 n_gpus_per_node。同时统一资源池定义，将原来从 teacher_model 读取的 n_gpus_per_node 提升到上层。
2. 教师管理拆分：在 verl/experimental/teacher_loop/teacher_model.py 中将原 TeacherModelManager 改造成仅管理单个教师，新增 MultiTeacherModelManager 类负责按 key 持有多个 TeacherModelManager 实例并分配资源池。TeacherModelManager 增加了 _validate_replica_node_alignment 校验，确保每个副本的子资源池不跨节点边界。
3. 请求路由重写：verl/experimental/teacher_loop/teacher_manager.py 中 AsyncTeacherLLMServerManager 不再继承 AsyncLLMServerManager，而是内部维护 server_managers: dict[str, AsyncLLMServerManager]，并通过 _resolve_teacher_key 根据样本 routing_key 选择对应服务器。采样参数获取函数也改为接收具体 DistillationTeacherModelConfig。
4. AgentLoopWorker 适配：verl/experimental/agent_loop/agent_loop.py 中 AgentLoopWorker.__init__ 参数 teacher_servers 和 teacher_load_balancer_handle 类型从列表变为字典，_compute_teacher_logprobs 方法传入 routing_key 并转发给 AsyncTeacherLLMServerManager。同时移除 colocate 模式的 wake_up/sleep 调用。

5. 训练入口与资源池初始化：verl/trainer/main_ppo_sync.py 和 main_ppo.py 中资源池分配逻辑改为从 distillation.n_gpus_per_node 和 distillation.nnodes 读取，并利用 MultiTeacherModelManager 实例化教师管理器。ray_trainer.py 和 fully_async_policy 的 agent_loop 导入对应更新。
6. 推理引擎统一：verl/workers/rollout/replica.py 以及 vLLM、SGLang、TRTLLM 的异步服务器均增加 name_suffix 参数，避免多教师下 Ray actor 名称冲突。
7. 示例脚本：新增 examples/on_policy_distillation_trainer/run_qwen3_mopd_gsm8k_geo3k.sh，展示为两个数据源配置不同教师（Qwen3-4B-Instruct 和 Qwen3-VL-4B-Instruct）蒸馏 Qwen3-VL-2B-Instruct 的用法。

关键文件：

- verl/experimental/teacher_loop/teacher_model.py（模块 教师管理；类别 source；类型 data-contract；符号 MultiTeacherModelManager, TeacherModelManager, _run_all, _initialize_load_balancer_handle）：核心数据结构变更，新增 MultiTeacherModelManager 类并重构 TeacherModelManager，包含资源池分割验证核心逻辑。
- verl/workers/config/distillation.py（模块 配置层；类别 source；类型 dependency-wiring；符号 DistillationTeacherModelConfig, DistillationConfig, per_replica_world_size, world_size）：配置数据契约核心文件，定义多教师场景下的 DistillationTeacherModelConfig 和 DistillationConfig，影响整个蒸馏流程的初始化。
- verl/experimental/teacher_loop/teacher_manager.py（模块 教师路由；类别 source；类型 dependency-wiring；符号 AsyncTeacherLLMServerManager, _resolve_teacher_key, _get_teacher_sampling_params）：AsyncTeacherLLMServerManager 重构为核心路由组件，不再继承 AsyncLLMServerManager，内部按教师 key 维护多个 server_manager。
- verl/experimental/agent_loop/agent_loop.py（模块 Agent 循环；类别 source；类型 core-logic；符号 AgentLoopWorker, _compute_teacher_logprobs）：AgentLoopWorker 初始化参数类型变更，并与 AsyncTeacherLLMServerManager 交互传递 routing_key，是路由逻辑的最终调用点。
- verl/trainer/main_ppo_sync.py（模块 训练器；类别 source；类型 dependency-wiring；符号 init_resource_pool_mgr, teacher_model_manager）：同步训练器入口，更新资源池初始化逻辑和教师管理器实例化方式，示范多教师与单教师的适配。
- verl/trainer/main_ppo.py（模块 训练器；类别 source；类型 core-logic；符号 init_resource_pool_mgr）：异步训练器入口，与 main_ppo_sync.py 同步的资源池变更。
- verl/workers/rollout/replica.py（模块 推理引擎；类别 source；类型 core-logic；符号 RolloutReplica）：RolloutReplica 基底类增加 name_suffix 参数，支持多教师场景下 Ray actor 名称不冲突。
- verl/workers/rollout/vllm_rollout/vllm_async_server.py（模块 推理引擎；类别 source；类型 core-logic；符号 VLLMAsyncServer）：vLLM 推理服务器适配 name_suffix 参数，避免多教师 Ray actor 名称冲突。

关键符号：MultiTeacherModelManager.init, TeacherModelManager._initialize_llm_servers, TeacherModelManager._validate_replica_node_alignment, AsyncTeacherLLMServerManager.init, AsyncTeacherLLMServerManager._resolve_teacher_key, AsyncTeacherLLMServerManager.compute_teacher_logprobs_single, AgentLoopWorker._compute_teacher_logprobs, DistillationTeacherModelConfig.per_replica_world_size, DistillationTeacherModelConfig.world_size, DistillationTeacherModelConfig.check_configured, DistillationConfig.post_init, AgentLoopWorker.init_resource_pool_mgr

关键源码片段

verl/experimental/teacher_loop/teacher_model.py

核心数据结构变更，新增 MultiTeacherModelManager 类并重构 TeacherModelManager，包含资源池分割验证核心逻辑。

```
# verl/experimental/teacher_loop/teacher_model.py（关键片段）
# MultiTeacherModelManager 负责将整个教师资源池按各教师所需的 GPU 数量切分，
# 并为每个教师实例化一个 TeacherModelManager。

@auto_await
async def _run_all(tasks: list[asyncio.Task]):
    await asyncio.gather(*tasks)

class MultiTeacherModelManager:
    """管理多个教师模型，每个教师有独立的 TeacherModelManager。"""

    def __init__(self, config: DictConfig, resource_pool: RayResourcePool):
        # 解析配置，获取每个教师的配置和所需总 GPU 数
        distillation_config: DistillationConfig = omega_conf_to_dataclass(config.distillation)
        total_world_size = 0
        teacher_configs: dict[str, DistillationTeacherModelConfig] = {}
        for key, cfg in distillation_config.teacher_models.items():
            cfg.check_configured()
            teacher_configs[key] = cfg
            total_world_size += cfg.world_size # num_replicas * per_replica_world_size

        # 校验资源池总大小是否匹配
        if resource_pool.world_size != total_world_size:
            raise ValueError(...)

        # 按每个教师的 world_size 依次切割资源池，构造子 TeacherModelManager
        self.teacher_managers: dict[str, TeacherModelManager] = {}
        start = 0
        for key, cfg in teacher_configs.items():
            sub_pool_size = cfg.world_size
            sub_resource_pool = RayResourcePool(
                bundles=resource_pool.bundles[start:start + sub_pool_size]
            )
```

```
            self.teacher_managers[key] = TeacherModelManager(
                distillation_config=distillation_config,
                teacher_model_config=cfg,
                resource_pool=sub_resource_pool,
            )
            start += sub_pool_size

class TeacherModelManager:
    """管理单个教师的推理服务器。"""

    def __init__(
        self,
        distillation_config: DistillationConfig,
        teacher_model_config: DistillationTeacherModelConfig,
        resource_pool: RayResourcePool,
    ):
        self.distillation_config = distillation_config
        self.teacher_model_config = teacher_model_config
        self.resource_pool = resource_pool
        self._initialize_llm_servers()
        self._initialize_load_balancer_handle()

    def _initialize_llm_servers(self):
        # 使用 teacher_model_config.per_replica_world_size 和 num_replicas 计算
        per_replica_world_size = self.teacher_model_config.per_replica_world_size
        num_replicas = self.teacher_model_config.num_replicas
        expected_pool_size = num_replicas * per_replica_world_size
        if self.resource_pool.world_size != expected_pool_size:
            raise ValueError(...)

        gpus_per_node = self.distillation_config.n_gpus_per_node
        name_suffix = (self.teacher_model_config.key or "").replace("/", "_")
        self.rollout_replicas = [
            rollout_replica_class(
                replica_rank=rank,
                config=rollout_config,
                model_config=model_config,
                gpus_per_node=gpus_per_node,
                is_teacher_model=True,
                name_suffix=name_suffix,
            ) for rank in range(num_replicas)
        ]
        split_resource_pools = split_resource_pool(self.resource_pool, split_size=per_replica_world_
        size)
        self._validate_replica_node_alignment(split_resource_pools, per_replica_world_size, gpus_
        per_node)
        _run_all([server.init_colocated(pool) for server, pool in zip(...)])
        # ... 设置 server_handles, server_addresses
```

```
    def _validate_replica_node_alignment(self, replica_pools, per_replica_world_size, gpus_per_
    node):
        """确保每个副本的子资源池不会跨节点过多，从而避免跨节点推理延迟。"""
        for i, pool in enumerate(replica_pools):
            expected_nodes = math.ceil(per_replica_world_size / gpus_per_node)
            # 检查 pool 中的 bundles 跨越的节点数是否符合预期
            actual_nodes = len(set(b.node_id for b in pool.bundles))
            if actual_nodes > expected_nodes:
                raise ValueError(
                    f"Replica {i} spans {actual_nodes} nodes, expected at most {expected_nodes}"
                )
```

verl/workers/config/distillation.py

配置数据契约核心文件，定义多教师场景下的 DistillationTeacherModelConfig 和 DistillationConfig，影响整个蒸馏流程的初始化。

```
# verl/workers/config/distillation.py（关键片段）
# 展示多教师配置模型的核心定义

@dataclass
class DistillationTeacherModelConfig(BaseConfig):
    """单个教师模型的配置。"""
    _mutable_fields = BaseConfig._mutable_fields | {"num_replicas", "key"}

    key: Optional[str] = None # 路由标识，如 "gsm8k"
    model_path: Optional[str] = None # 教师模型路径
    inference: RolloutConfig = field(default_factory=RolloutConfig)
    num_replicas: Optional[int] = 0 # 该教师启动的推理副本数

    @property
    def per_replica_world_size(self) -> int:
        """每个副本需要的 GPU 数（TP * DP * PP）。"""
        return (
            self.inference.tensor_model_parallel_size
            * self.inference.data_parallel_size
            * self.inference.pipeline_model_parallel_size
        )

    @property
    def world_size(self) -> int:
        """该教师占用的总 GPU 数。"""
        return self.num_replicas * self.per_replica_world_size

    def check_configured(self):
        if self.model_path is None:
            raise ValueError("model_path must be specified")
        if self.key is None:
            raise ValueError("key must be specified")
        if self.num_replicas is None or self.num_replicas <= 0:
```

```
            raise ValueError("num_replicas must be set > 0")

@dataclass
class DistillationConfig(BaseConfig):
    """顶层蒸馏配置，支持多教师。"""
    _mutable_fields = BaseConfig._mutable_fields | {"teacher_models"}

    enabled: bool = False
    n_gpus_per_node: int = 8 # 每节点 GPU 数，用于资源池分配
    nnodes: int = 1
    teacher_models: dict[str, DistillationTeacherModelConfig] = field(default_factory=dict)
    teacher_key: str = "data_source" # 用作路由的数据字段名
    distillation_loss: DistillationLossConfig = field(default_factory=DistillationLossConfig)

    def __post_init__(self):
        # 将 teacher_models 重新按 key 索引（来自 config 的键）
        rekeyed = {}
        for config_key, cfg in self.teacher_models.items():
            cfg.key = cfg.key or config_key # 若未显式设置 key，使用配置字典的键
            rekeyed[cfg.key] = cfg
        self.teacher_models = rekeyed
```

verl/experimental/teacher_loop/teacher_manager.py

AsyncTeacherLLMServerManager 重构为核心路由组件，不再继承 AsyncLLMServerManager，内部按教师 key 维护多个 server_manager。

```
# verl/experimental/teacher_loop/teacher_manager.py（关键片段）
# AsyncTeacherLLMServerManager 内部持有多个 AsyncLLMServerManager，每个对应一个教师

class AsyncTeacherLLMServerManager:
    """多教师路由管理器，内部持有多个 AsyncLLMServerManager。"""

    def __init__(
        self,
        config: DictConfig,
        servers: dict[str, list[tuple[str, ray.actor.ActorHandle]]],
        load_balancer_handle: dict[str, ray.actor.ActorHandle],
    ):
        self.distillation_config: DistillationConfig = omega_conf_to_dataclass(config.distillation)
        self.distillation_loss_config = self.distillation_config.distillation_loss
        self.teacher_key = self.distillation_config.teacher_key
        self.teacher_model_configs: dict[str, DistillationTeacherModelConfig] = \
            self.distillation_config.teacher_models

        # 校验 servers 和 load_balancer_handle 的键与教师配置键一致
        expected = set(self.teacher_model_configs)
        assert set(servers.keys()) == expected, ...
        assert set(load_balancer_handle.keys()) == expected, ...
```

```
    # 为每个教师创建一个 AsyncLLMServerManager
    self.server_managers: dict[str, AsyncLLMServerManager] = {
        key: AsyncLLMServerManager(
            config=config,
            servers=servers[key],
            load_balancer_handle=load_balancer_handle[key],
        )
        for key in self.teacher_model_configs
    }

def _resolve_teacher_key(self, routing_key: Optional[str]) -> str:
    """根据样本的路由键确定使用哪个教师。单教师场景无需路由键。"""
    if len(self.teacher_model_configs) == 1:
        return next(iter(self.teacher_model_configs))
    if routing_key is None:
        raise ValueError(
            f"Multi-teacher requires routing key via `{self.teacher_key}`, "
            f"but got None. Configured teachers: {sorted(self.teacher_model_configs)}"
        )
    if routing_key not in self.teacher_model_configs:
        raise ValueError(f"No teacher for routing key {routing_key!r}.")
    return routing_key

async def compute_teacher_logprobs_single(
    self,
    sequence_ids: list[int],
    multi_modal_data: Optional[dict] = None,
    routing_key: Optional[str] = None,
) -> tuple[torch.Tensor, torch.Tensor]:
    teacher_key = self._resolve_teacher_key(routing_key)
    teacher_config = self.teacher_model_configs[teacher_key]
    server_manager = self.server_managers[teacher_key]
    # 使用该教师专用的采样参数
    sampling_params = _get_teacher_sampling_params(teacher_config, self.distillation_loss_
    config)
    teacher_output = await server_manager.generate(
        request_id=uuid4().hex,
        prompt_ids=sequence_ids,
        sampling_params=sampling_params,
        image_data=multi_modal_data.get("images"),
        video_data=multi_modal_data.get("videos"),
    )
    teacher_ids = torch.tensor(teacher_output.extra_fields["prompt_ids"], dtype=torch.int32)
    teacher_logprobs = torch.tensor(teacher_output.extra_fields["prompt_logprobs"])
    return teacher_ids, teacher_logprobs
```

评论区精华

- 断言过于严格：gemini-code-assist 指出 assert "teacher_model" in self.teacher_models 在用户完全覆盖默认字典时会导致崩溃，建议改为优雅检查。作者在后续提交中可能已调整（当前提交记录显示有 "Re-key" 提交）。
- 多节点限制不当：gemini-code-assist 认为 nnodes=1 的限制过于严格，而 _validate_replica_node_alignment 已能保证节点边界正确，建议移除。作者在提交 "Drop coarse nnodes=1 guard" 中移除了该限制。
- 未使用的 Router 进程：gemini-code-assist 指出 _initialize_router 启动的 sidecar 从未被使用（实际使用 Ray LoadBalancer），造成资源浪费。作者在提交 "Drop unused teacher-side router sidecar" 中清理了该代码。
- num_replicas 命名：wuxibin89 建议将 world_size 改为 num_replicas 更直观，因为用户只需指定副本数而无需计算总 GPU。作者接受了建议，在配置中改为 num_replicas。
- 配置注释不足：wuxibin89 要求对复杂的 _resolve_teacher_models 逻辑添加注释，作者在后续提交中补充了文档字符串。
  - 断言 teacher_model 导致的崩溃风险 (correctness): 作者在后续提交中通过重写 teacher_models 的重索引逻辑解决了该问题（见 "Re-key" 系列提交）。
  - 多节点限制 nnodes=1 过于严格 (design): 作者在提交 "Drop coarse nnodes=1 guard" 中移除了这一限制。
  - 未使用的 Router sidecar 进程 (performance): 作者在提交 "Drop unused teacher-side router sidecar" 中删除了相关方法。
  - num_replicas 命名字段选择 (design): 作者接受建议，在 DistillationTeacherModelConfig 中改用 num_replicas 字段。
  - 配置解析逻辑缺少说明注释 (documentation): 作者在后续提交中补充了 docstring 和配置文件的注释。

风险与影响

- 风险：
  - 配置复杂性风险：多教师配置引入 teacher_models 字典和 teacher_key，用户需要正确设置每个教师的 num_replicas、per_replica_world_size 等参数，错误配置可能导致资源分配失败或 GPU 浪费。涉及文件 verl/workers/config/distillation.py。
  - 资源池对齐校验遗漏：_validate_replica_node_alignment 在特定节点拓扑下可能误报或漏报，若未充分测试多节点场景，可能导致初始化静默失败或跨节点副本挂起。涉及 verl/experimental/teacher_loop/teacher_model.py。
  - 路由逻辑正确性：_resolve_teacher_key 的行为默认使用单教师，当数据集不含有效 routing_key 时可能抛出 ValueError，但生产环境可能希望有默认教师降级。涉及 verl/experimental/teacher_loop/teacher_manager.py。
  - 缺少测试覆盖：本次 PR 包含大量核心逻辑变更，但未发现对应的单元测试或集成测试，尤其在多教师路由、资源池分割、节点对齐等关键路径上缺少验证。
- 影响：
  - 用户影响：用户现在可以为不同数据源配置不同的教师模型，从而让学生从多个专家模型中蒸馏知识。但需要理解新的配置字段（teacher_models、teacher_key、num_replicas）

，旧配置（teacher_model、n_gpus_per_node、nnodes）不再兼容。

- 系统影响：每个教师模型需要独立的 GPU 资源池，总体 GPU 消耗增加。同时，推理服务器的 Ray actor 名称通过 name_suffix 避免冲突，但多教师场景下资源管理更复杂。
- 团队影响：代码从 TeacherModelManager 拆分为 MultiTeacherModelManager + TeacherModelManager，职责更清晰，但后续维护者需要理解两层管理器的交互。
- 风险标记：缺少测试覆盖，配置复杂性，资源分配错误，多节点部署风险，路由降级未定义

关联脉络

- PR #6358 [doc] chore: OPD docs: 本 PR 的示例脚本 run_qwen3_mopd_gsm8k_geo3k.sh 在 #6358 中被进一步文档化和规范化，同时 #6358 的回调脚本也用到了本 PR 的多教师能力。
- PR #6350 [fsdp] fix: emit distillation outputs in use_remove_padding=False path (#6293): 修复了 FSDP 蒸馏输出路径的 bug，与本 PR 的多教师蒸馏运行时直接相关。
- PR #6345 [fsdp] fix: build no-padding attention mask from input ids: 影响 FSDP 下蒸馏计算中 mask 的正确性，多教师场景可能同样受影响。
- PR #6334 [fsdp, ckpt] fix: drop tied target keys before HF save_pretrained: 修改了检查点保存逻辑，教师模型权重保存路径可能与之交互。