PR #28671 完整报告

sgl-project/sglang

AutoWeightLoader support Sglang native models 1: demo

合并时间：2026-07-22 05:32

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28671

执行摘要

- 一句话：AutoWeightLoader v2 demo: Qwen2/Llama 集中式权重加载
- 推荐动作：值得精读，因为这是 weight loader 重构的基石。重点关注 StackedParamsDispatch 的设计、AutoWeightsLoader walker 与子模块 load_weights 的协作模式，以及 RemapRegistry 的注册机制。对于计划参与模型迁移的工程师，建议深入理解该 PR 的接口契约。

功能与动机

现有 165+ 模型文件各自实现 load_weights，重复 boilerplate 且修复常遗漏（如 FP8 kv-scale remapping、PP 支持）。此 PR 开始集中化，降低维护成本，确保跨模型一致性。参见 issue #24703。

实现拆解

1. 新增中央加载模块：在 python/sglang/srt/model_loader/auto_loader.py 中定义 StackedParamsDispatch（msgspec.Struct，管理 fused qkv/gate_up 的 shard 路由）、filter_pp_weights（PP 层过滤）、RemapRegistry（架构特定 weight remap，带 register_weight_remap 装饰器和 get_weight_remap 访问器）。预置 STANDARD_QKV_MAPPING、STANDARD_GATE_UP_MAPPING 等实例。
2. 模型子模块加载器：分别为 Qwen2MLP、Qwen2Attention、LlamaMLP、LlamaAttention 添加 load_weights 方法，从 auto_loader 导入标准 mapping 并利用 try_load 处理融合参数，未匹配参数走直接加载（weight_loader）。
3. 顶层模型切换：在 Qwen2Model 和 LlamaModel 中修改 load_weights，根据环境变量 SGLANG_ENABLE_WEIGHT_LOADER_V2 选择传统 _legacy_load_weights 或新 _load_weights_v2。新路径使用 AutoWeightsLoader（models/utils.py 中的 walker）配合 filter_pp_weights、RemapRegistry 完成加载。
4. 环境变量：在 python/sglang/srt/environ.py 添加 SGLANG_ENABLE_WEIGHT_LOADER_V2（默认 False），实现逐步切出。
5. 测试覆盖：新增 test/registered/model_loading/test_weight_loader_v2_e2e.py（SRTRunner 端到端验证 Qwen2 v1/v2 生成一致 + transformers 实现可加载）和 test/manual/test_weight_loader_v2_equiv.py（手动 verify state_dict 逐参数相等）。

关键文件：

- python/sglang/srt/model_loader/auto_loader.py（模块 权重加载器；类别 source；类型 data-contract；符号 StackedParamsDispatch, try_load, filter_pp_weights, register_weight_remap）：核心新文件，定义权重加载集中抽象：StackedParamsDispatch、RemapRegistry、filter_pp_weights 等，是整个 PR 的设计核心。
- python/sglang/srt/models/qwen2.py（模块 模型实现；类别 source；类型 data-contract；符号 load_weights, _legacy_load_weights, _load_weights_v2）：演示模型之一：子模块新增 load_weights 使用 STANDARD_GATE_UP_MAPPING 和 STANDARD_QKV_MAPPING，顶层新增 v2 切换逻辑。
- python/sglang/srt/models/llama.py（模块 模型实现；类别 source；类型 data-contract；符号 load_weights, _legacy_load_weights, _load_weights_v2）：第二个演示模型，与 Qwen2 类似但额外使用 RemapRegistry（FP8 后缀归一化），展示 remap 机制。
- test/registered/model_loading/test_weight_loader_v2_e2e.py（模块 集成测试；类别 test；类型 test-coverage；符号 TestWeightLoaderV2E2E, setUpClass, _runner_kwargs, test_qwen2_native_v1_v2_generation_match）：端到端 CI 测试，验证 Qwen2 在 v1/v2 下的生成结果一致，是效果正确性的主要保障。
- test/manual/test_weight_loader_v2_equiv.py（模块 手动测试；类别 test；类型 test-coverage；符号 _init_model_parallel, _load_qwen2_native, _state_dict_cpu, TestWeightLoaderV2Equiv）：手动验证 state_dict 级别等价，确保每个参数值精确一致，弥补 e2e 测试的数值近似。
- python/sglang/srt/environ.py（模块 配置项；类别 source；类型 core-logic）：添加环境变量 SGLANG_ENABLE_WEIGHT_LOADER_V2，作为功能开关，控制新旧加载路径。

关键符号：StackedParamsDispatch.try_load, filter_pp_weights, register_weight_remap, get_weight_remap, Qwen2Model._load_weights_v2, LlamaModel._load_weights_v2, Qwen2MLP.load_weights, Qwen2Attention.load_weights, LlamaMLP.load_weights, LlamaAttention.load_weights

关键源码片段

python/sglang/srt/model_loader/auto_loader.py

核心新文件，定义权重加载集中抽象：StackedParamsDispatch、RemapRegistry、filter_pp_weights 等，是整个 PR 的设计核心。

```
class StackedParamsDispatch(msgspec.Struct, frozen=True):
    """集中式 stacked 参数加载，用于融合线性层。

    处理从 checkpoint 名称（q_proj, k_proj, v_proj, gate_proj, up_proj）到运行时
    融合参数（qkv_proj, gate_up_proj）的映射，并分配正确的 shard_id。
    量化逻辑完全由 param.weight_loader 处理，此类只负责路由。
    """

    # (fused_param_name, checkpoint_source_name, shard_id)
    mappings: tuple[tuple[str, str, Union[int, str]], ...] = ()
```

```
    def try_load(
        self,
        name: str,
        tensor: torch.Tensor,
        params_dict: dict[str, Parameter],
    ) -> str | None:
        """尝试通过 stacked mapping 加载权重。

        返回已加载的运行时参数名称（匹配则加载）或 target 名称（用于跳过跟踪），
        如果目标参数不存在（如可选的 bias），则返回 target 名称；无匹配返回 None。
        """
        for fused_name, source_name, shard_id in self.mappings:
            if source_name not in name:
                continue
            target = name.replace(source_name, fused_name)
            param = params_dict.get(target)
            if param is None:
                # 参数不存在——例如 GPTQ bias
                return target
            param.weight_loader(param, tensor, shard_id)
            return target
        return None


# 预置最常用的 decoder 模式
STANDARD_QKV_MAPPING = StackedParamsDispatch(
    mappings=(
        ("qkv_proj", "q_proj", "q"),
        ("qkv_proj", "k_proj", "k"),
        ("qkv_proj", "v_proj", "v"),
    )
)

STANDARD_GATE_UP_MAPPING = StackedParamsDispatch(
    mappings=(
        ("gate_up_proj", "gate_proj", 0),
        ("gate_up_proj", "up_proj", 1),
    )
)
```

python/sglang/srt/models/qwen2.py

演示模型之一：子模块新增 load_weights 使用 STANDARD_GATE_UP_MAPPING 和 STANDARD_QKV_MAPPING，顶层新增 v2 切换逻辑。

```
class Qwen2Model(nn.Module):
    # ...
    def load_weights(self, weights: Iterable[Tuple[str, torch.Tensor]]):
        from sglang.srt.environ import envs
```

```
        if envs.SGLANG_ENABLE_WEIGHT_LOADER_V2.get():
            return self._load_weights_v2(weights)
        return self._legacy_load_weights(weights)

    def _load_weights_v2(self, weights):
        """AutoWeightsLoader-based weight loading."""
        from sglang.srt.model_loader.auto_loader import (
            AutoWeightsLoader,
            filter_pp_weights,
        )

        # PP 层过滤
        if hasattr(self.model, "start_layer"):
            weights = filter_pp_weights(
                weights, self.model.start_layer, self.model.end_layer
            )

        skip_prefixes = []
        if self.config.tie_word_embeddings:
            skip_prefixes.append("lm_head.")

        loader = AutoWeightsLoader(self, skip_prefixes=skip_prefixes)
        loaded = loader.load_weights(weights)

        # tie weight 同步
        if self.config.tie_word_embeddings:
            params_dict = dict(self.named_parameters())
            if "lm_head.weight" in params_dict:
                embed = dict(self.model.named_parameters()).get("embed_tokens.weight")
                if embed is not None:
                    params_dict["lm_head.weight"].weight_loader(
                        params_dict["lm_head.weight"], embed.data
                    )
        return loaded
```

评论区精华

- b8zhong 建议使用 msgspec.Struct：要求 StackedParamsDispatch 改为 msgspec.Struct 以符合代码库惯例。作者采纳。
- WeightRemapRegistry 演示不充分：b8zhong 希望此 PR 在另一个模型上演示 remap 用法（如 Llama FP8 remap）。作者后续在 Llama 路径中集成了 get_weight_remap。
- design.md 应移出代码库：Fridge003 建议将设计文档迁至单独 issue。作者随后删除本地 design.md，将链接指向 issue #31051。
- 手动测试位置争议：Fridge003 提议将 test_weight_loader_v2_equiv.py 移到 registered 测试；作者保留为手动测试，理由是该测试较重且检查所有参数 equality，与轻量级 e2e 互补。

- 单元测试方案调整：初始版本包含 test_auto_loader.py 单元测试，review 后被替换为 *e2e* 和手动测试。
  - StackedParamsDispatch 使用 msgspec.Struct (design): 作者修改 StackedParamsDispatch 继承 msgspec.Struct。
  - WeightRemapRegistry 的演示 (design): 作者在 Llama v2 路径中集成了 get_weight_remap。
  - design.md 移出代码库 (documentation): 作者删除 design.md 并将链接指向 issue #31051。
  - 手动测试位置 (testing): 作者保留为手动测试，因为测试开销较大且专注于 state_dict 精确相等。
  - 单元测试替换为 *e2e* (testing): 删除 test_auto_loader.py，改用 test_weight_loader_v2_*e2e*.py 和 test_weight_loader_v2_equiv.py。

风险与影响

- 风险：
  1. 正确性回归：新路径与旧路径可能产生微小的数值差异（如浮点顺序、量化路线的 weight_loader 行为差异），虽然 *e2e* 测试 tight tolerance 覆盖 Qwen2，但 Llama 等其他模型未完全验证。
  2. 量化兼容性：当前仅演示非量化场景；GPTQ/AWQ 等量化路径的 weight_loader 在 try_load 中可能需处理 shard_id 表示（字符串 vs 整数），msgspec.Struct 的 frozen 属性可能限制后续扩展。
  3. PP 支持边缘情况：filter_pp_weights 依赖 start_layer/end_layer，与 AutoWeightsLoader walker 的交互可能遗漏新层类型。
  4. 性能开销：params_dict 的构建在子模块 load_weights 中可能略高于一次性遍历，但影响较小。
- 影响：
  - 用户：无直接影响（默认关闭）。未来迁移模型后需测试兼容性。
  - 系统：引入新的加载抽象和切换开关，降低后续 weight loader 修复的跨模型重复工作。
  - 团队：提供清晰的迁移路径（PR1–PR9），后续开发者可参照此模式逐步迁移剩余模型。
  - 风险标记：核心权重加载路径变更，量化兼容性未验证，仅覆盖两个模型，PP 层过滤正确性依赖

关联脉络

- 暂无明显关联 PR