# PR #24431 完整报告

sgl-project/sglang

[diffusion] fix: fix diffusion FSDP sharding

合并时间：2026-05-06 14:55

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24431

## 执行摘要

- 一句话：集中扩散模型 FSDP 分片条件并修复权重加载与包装器兼容性
- 推荐动作：值得精读，尤其是 FSDP 分片条件的集中设计和通用回退机制。设计决策（如基于类名和通用编号块的自动分片）具有借鉴意义。但需关注回归问题的修复进展。

## 功能与动机

扩散模型使用 FSDP 进行分片推理时，不同模型的分片条件分散在各自配置中，难以维护且容易遗漏。FSDP 与张量并行（TP）的权重加载存在冲突，需要先加载预分片权重再分发。FSDP 包装后的模型无法通过简单 getattr 访问内部属性。本 PR 集中分片条件、修复加载逻辑并添加通用回退机制，以降低维护成本并提高模型兼容性。

## 实现拆解

1. 集中 FSDP 分片条件函数：新增 configs/models/fsdp.py，定义 is_module_list_entry、is_layer、is_block、is_double_block 等系列函数，统一通过模块名称判断分片边界。各模型配置（如 hunyuanvideo、gemma2 等）删除内联函数并改为导入集中函数。
2. 重构 FSDP 加载器：修改 runtime/loader/fsdp_load.py，新增 _get_param_for_weight_loading 函数以支持带 weight_loader 的参数；新增 _is_common_numbered_block 和 _resolve_fsdp_shard_conditions 函数，提供基于模块类名或通用编号块的回退分片策略；在 maybe_load_fsdp_model 中先保存所有带 weight_loader 的参数，再执行 FSDP 包装。
3. 增强包装兼容性：修改 runtime/pipelines_core/stages/denoising.py，新增 _get_transformer_attr 方法递归搜索 FSDP 包装模块（_fsdp_wrapped_module、module、_orig_mod）以获取属性；将 prepare_extra_func_kwargs 重构为 _get_extra_func_kwarg_names 并添加 _extra_func_kwarg_names_cache 缓存，避免每步签名检查的开销。
4. 更新性能基线：更新 test/registered/fp8/fp8_perf_baselines.txt 以反映当前 FSDP 推理性能。
5. 测试与配置调整：修改多个模型配置文件的导入，确保兼容集中式函数。

关键文件：

- python/sglang/multimodal_gen/configs/models/fsdp.py（模块 扩散模型；类别 source；类型 data-contract；符号 is_module_list_entry, is_module_list_entry_in, is_layer,

is_block）：新增的核心文件，集中定义了所有 FSDP 分片条件函数，是整个 PR 的基石。

- python/sglang/multimodal_gen/runtime/loader/fsdp_load.py（模块 加载器；类别 source；类型 dependency-wiring；符号 _get_param_for_weight_loading, _make_class_name_shard_condition, _is_common_numbered_block, _resolve_fsdp_shard_conditions）：修改了权重加载逻辑，新增预分片参数保存、回退分片条件函数，修复 FSDP+TP 兼容性。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/denoising.py（模块 降噪阶段；类别 source；类型 core-logic；符号 _get_transformer_attr, prepare_extra_func_kwargs, _get_extra_func_kwarg_names）：修改了降噪阶段，添加包装模块属性访问和 kwarg 签名缓存，提升兼容性和性能。
- python/sglang/multimodal_gen/configs/models/dits/hunyuanvideo.py（模块 扩散模型；类别 source；类型 data-contract；符号 is_double_block, is_single_block, is_refiner_block, is_txt_in）：删除了内联分片条件函数，改为从集中文件导入，体现集中化模式。
- python/sglang/multimodal_gen/configs/models/encoders/gemma2.py（模块 编码器；类别 source；类型 data-contract；符号 _is_transformer_layer, _is_embeddings, _is_final_norm）：类似 hunyuanvideo，删除内联条件改为导入集中函数，代表所有编码器的统一变更。

关键符号：is_module_list_entry, is_module_list_entry_in, _get_param_for_weight_loading, _resolve_fsdp_shard_conditions, _is_common_numbered_block, _get_transformer_attr, _get_extra_func_kwarg_names

## 关键源码片段

### python/sglang/multimodal_gen/configs/models/fsdp.py

新增的核心文件，集中定义了所有 FSDP 分片条件函数，是整个 PR 的基石。

```
# SPDX-License-Identifier: Apache-2.0


def is_module_list_entry(name: str, container_name: str) -> bool:
    # 匹配直接属于某容器的子模块（数字索引），排除其内部子模块
    parts = name.split(".")
    return len(parts) >= 2 and parts[-2] == container_name and parts[-1].isdigit()


def is_module_list_entry_in(name: str, container_names: tuple[str, ...]) -> bool:
    # 匹配直接属于多个候选容器之一的子模块
    parts = name.split(".")
    return len(parts) >= 2 and parts[-2] in container_names and parts[-1].isdigit()


def is_layer(name: str, module: object) -> bool:
    # 匹配 "layers" 容器下的直接子模块
    return is_module_list_entry(name, "layers")
```

```
def is_block(name: str, module: object) -> bool:
    # 匹配 "blocks" 容器下的直接子模块
    return is_module_list_entry(name, "blocks")


def is_double_block(name: str, module: object) -> bool:
    # 匹配 "double_blocks" 容器下的直接子模块（用于 HunyuanVideo 等模型）
    return is_module_list_entry(name, "double_blocks")


def is_single_block(name: str, module: object) -> bool:
    # 匹配 "single_blocks" 容器下的直接子模块
    return is_module_list_entry(name, "single_blocks")


def is_refiner_block(name: str, module: object) -> bool:
    # 匹配 "refiner_blocks" 容器下的直接子模块
    return is_module_list_entry(name, "refiner_blocks")
```

## python/sglang/multimodal_gen/runtime/loader/fsdp_load.py

修改了权重加载逻辑，新增预分片参数保存、回退分片条件函数，修复 FSDP+TP 兼容性。

```
def _get_param_for_weight_loading(
    model: torch.nn.Module,
    param_dict: dict[str, torch.nn.Parameter],
    param_name: str,
) -> torch.nn.Parameter | None:
    # 优先返回带 weight_loader 的参数，用于自定义加载逻辑
    actual_param = param_dict.get(param_name)
    if actual_param is not None and getattr(actual_param, "weight_loader", None):
        return actual_param

    # 查找预 FSDP 保存的带 weight_loader 的参数（在包装前保存的）
    pre_fsdp_weight_loader_params = getattr(model, "_pre_fsdp_weight_loader_params", {})
    pre_fsdp_param = pre_fsdp_weight_loader_params.get(param_name)
    if pre_fsdp_param is not None:
        return pre_fsdp_param

    return actual_param


def _make_class_name_shard_condition(class_names: set[str]):
    # 根据模块类名创建分片条件
    def shard_condition(n: str, m: nn.Module) -> bool:
        return type(m).__name__ in class_names
    return shard_condition


def _is_common_numbered_block(n: str, m: nn.Module) -> bool:
```

```
    # 通用编号块匹配：识别常见容器名下的数字索引子模块
    return is_module_list_entry_in(
        n,
        (
            "blocks",
            "layers",
            "double_blocks",
            "single_blocks",
            "refiner_blocks",
            "noise_refiner",
            "context_refiner",
            "transformer_blocks",
            "single_transformer_blocks",
        ),
    )


def _resolve_fsdp_shard_conditions(
    model: torch.nn.Module,
    fsdp_shard_conditions: list[Callable[[str, nn.Module], bool]] | None,
) -> tuple[list[Callable[[str, nn.Module], bool]], str]:
    # 优先级：显式条件 > 基于模型 _repeated_blocks/_no_split_modules 的类名条件 >
    通用编号块条件
    if fsdp_shard_conditions:
        return fsdp_shard_conditions, "explicit"

    block_class_names = set(getattr(model, "_repeated_blocks", []) or [])
    block_class_names.update(getattr(model, "_no_split_modules", []) or [])
    if block_class_names:
        return [_make_class_name_shard_condition(block_class_names)], "block-class"

    return [_is_common_numbered_block], "common-numbered-block"
```

## python/sglang/multimodal_gen/runtime/pipelines_core/stages/denoising.py

修改了降噪阶段，添加包装模块属性访问和 kwarg 签名缓存，提升兼容性和性能。

```
def _get_transformer_attr(self, name: str) -> Any:
    # 递归搜索 transformer 及其 FSDP 包装属性，找到第一个非 None 的值
    seen: set[int] = set()
    stack = [self.transformer]
    while stack:
        module = stack.pop()
        if module is None or id(module) in seen:
            continue
        seen.add(id(module))

        value = getattr(module, name, None)
        if value is not None:
            return value
```

```
        # 遍历常见的 FSDP 包装属性名称
        for wrapper_attr in ("_fsdp_wrapped_module", "module", "_orig_mod"):
            wrapped = getattr(module, wrapper_attr, None)
            if wrapped is not None:
                stack.append(wrapped)
    return None

def _get_extra_func_kwarg_names(self, func) -> tuple[bool, frozenset[str]]:
    # 缓存函数的可变参标志和参数名集合，避免每次调用都反射签名
    import functools
    # 处理 cache-dit 的 partial 包装
    if isinstance(func, functools.partial):
        func = func.func
    target_func = inspect.unwrap(func)
    cache_target = (
        target_func.__func__ if inspect.ismethod(target_func) else target_func
    )
    cache_key = id(cache_target)
    cached = self._extra_func_kwarg_names_cache.get(cache_key)
    if cached is not None:
        return cached

    params = inspect.signature(target_func).parameters
    result = (
        any(param.kind == inspect.Parameter.VAR_KEYWORD for param in params.values()),
        frozenset(params),
    )
    self._extra_func_kwarg_names_cache[cache_key] = result
    return result
```

## 评论区精华

唯一的评论来自 gemini-code-assist[bot]，指出 fsdp_load.py 中一处 requires_grad 赋值是冗余的（因为 _make_param_like 已设置 requires_grad=False），建议简化。作者未在评论中回复，该行在最终版本中已保留，可能认为该冗余在上下文中无害。

- 移除冗余 requires_grad 赋值 (style): 未采纳该建议，该行在最终版本中保留，可能认为其无害或为保险起见。

## 风险与影响

- 风险：
  1. 回归风险：评论指出模型 Efficient-Large-Model/Sana_600M_512px_diffusers 在该 PR 后无法运行，可能源于分片条件或加载逻辑变化。作者已同意修复，但合并前需确保该模型正常。
  2. 集中函数可能改变原有分片边界，影响性能或正确性。

3. 缓存机制可能引入陈旧数据，需确保缓存键正确更新。 - 影响：影响所有使用 FSDP 推理的扩散模型，包括 HunyuanVideo、Gemma、LLaMA、T5 等编码器 / 解码器。变更涉及 29 个文件，但核心逻辑集中在加载器和降噪阶段。对于未使用 FSDP 的模型无影响。 - 风险标记：回归风险，模型兼容性，缓存一致性

# 关联脉络

- 暂无明显关联 PR