PR #35668 完整报告

sgl-project/sglang

[diffusion] feat: add weight source reader

合并时间：2026-08-20 18:37

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35668

执行摘要

- 一句话：新增权重读取后端协议，重构 checkpoint 加载选择逻辑
- 推荐动作：值得精读。它示范了如何用协议 + 能力标记取代布尔参数：能力放在声明者身上而非调用点，选择逻辑集中且 fail-fast。PR Notes 中的带宽实测（pinned 匿名 13.09 GB/s、pageable 匿名 8.84 GB/s、cache-resident 文件映射 8.5 GB/s；per-tensor 拷贝 13.41 GB/s vs 合并拷贝 13.39 GB/s）表明，未来把 offload 的 host 缓冲迁移到文件映射源是可行且几乎无损的——retains_file_mapping 就是为这个 follow-up 声明的地基。建议关注后续 host copy 与 offload 相关 PR。

功能与动机

PR body 明确指出原实现的三个问题：其一，读取 checkpoint 只是一个布尔值，safetensors_weights_iterator 接受 use_runai_model_streamer: bool | None 并在函数内联分支，调用点还硬编码了一条规则「key_filter 存在时强制关闭 streamer」；其二，这条注释描述的是 Run:ai streamer 的属性（materialize 所有 tensor 后才 yield，无法在加载时跳过 partition），却放在调用点，每个未来调用者都要重新发现它；其三，两条路径的差异不止速度——safe_open 通过 mmap 映射文件，产出的 CPU tensor 是文件视图，而 streamer 拷贝进匿名内存，「Nothing named that difference, which makes it invisible to anyone reasoning about host memory」。这正是引入 retains_file_mapping 能力标记的动机。

实现拆解

1. 定义协议与能力标记：新建 python/sglang/multimodal_gen/runtime/loader/weight_readers/base.py，用 @runtime_checkable 声明 WeightReader Protocol，包含 name、supports_key_filter、retains_file_mapping 三个 ClassVar 能力标记，以及 is_available() 与 iter_weights() 两个接口。协议把「能否跳过 key」「产出的 tensor 是否保留文件映射」这两个决定正确性与内存行为的属性显式化，取代原来散落在调用点的隐式规则。
2. 实现两个读取后端：新建 runai_streamer.py 的 RunaiStreamerReader（supports_key_filter=False、retains_file_mapping=False，is_available() 依赖 runai_model_streamer 是否安装，to_cpu 与 clone_tensors 决定是否克隆）与 safetensors_mmap.py 的 SafetensorsMmapReader（基于 safe_open，supports_key_filter=True、retains_file_mapping=True，始终可用）。每个后端把原来 weight_utils.py 内联分支中的读取循环（streamer 的 stream_files/get_tensors，mmap

的逐 shard get_tensor）收进各自的 iter_weights。

3. 集中选择逻辑：新建 weight_readers/__init__.py，select_weight_reader() 统一决策——显式 requested 优先，未知名字直接抛 ValueError（fail-fast，而不是静默回退）；未指定时读 envs.SGLANG_USE_RUNAI_MODEL_STREAMER 且要求 streamer 可用；选中后做两级兜底：未安装则回退到 FALLBACK_READER，调用方需要 key_filter 而选中项不支持时也回退（因为读完整份 checkpoint 再丢弃大部分比慢速读取所需部分更糟）。available_reader_names() 提供可发现性。
4. 改造加载入口：修改 weight_utils.py 的 safetensors_weights_iterator()，删除内联布尔分支，把旧布尔参数映射为 reader 名字（True → runai_streamer、False → safetensors、None → 环境决定），改为 backend = select_weight_reader(requested=requested, needs_key_filter=key_filter is not None) 后 yield from backend.iter_weights(...)。文件完整性预检（_scan_safetensors_files 的 corrupted/duplicate 检查）仍保留在调用方，职责分离清晰；克隆语义由原 clone_streamed_tensors 映射为协议的 clone_tensors 参数。HAS_RUNAI_MODEL_STREAMER 常量移到 runai_streamer.py 并从该模块重新导入。
5. 测试配套：新建 test_weight_readers.py，10 个单测覆盖两类行为——能力标记（streamer 不能跳过 key、只有 mmap 后端保留可回收页、fallback 始终可用）与选择策略（显式请求被尊重、未知名报错而非静默回退、key_filter 跳过不支持的后端、不可用后端回退、环境变量决策）。作者用 revert 本改动复跑 4 个既有失败（realtime/、sana_wm/）证明失败与本 PR 无关。

关键文件：

- python/sglang/multimodal_gen/runtime/loader/weight_readers/base.py（模块 权重加载；类别 source；类型 core-logic；符号 WeightReader, is_available, iter_weights）：定义 WeightReader 协议与两个能力标记，是整个重构的设计核心，把「能否跳过 key」「是否保留文件映射」从隐式规则变为显式契约。
- python/sglang/multimodal_gen/runtime/loader/weight_readers/__init__.py（模块 权重加载；类别 source；类型 core-logic；符号 select_weight_reader, available_reader_names）：select_weight_reader 集中了全部后端选择决策，含显式请求优先、fail-fast 报错与两级回退，是未来扩展新读取后端的唯一注册点。
- python/sglang/multimodal_gen/runtime/loader/weight_readers/runai_streamer.py（模块 权重加载；类别 source；类型 core-logic；符号 RunaiStreamerReader, is_available, iter_weights）：把 Run:ai streamer 读取逻辑从 weight_utils 的内联分支迁出，并通过 is_available 表达可选依赖；supports_key_filter=False 解释了原调用点那条硬编码规则。
- python/sglang/multimodal_gen/runtime/loader/weight_readers/safetensors_mmap.py（模块 权重加载；类别 source；类型 core-logic；符号 SafetensorsMmapReader, is_available, iter_weights）：safe_open 读取逻辑的封装，作为始终可用的 fallback；retains_file_mapping=True 是后续 host 内存回收方案的关键声明，且 supports_key_filter=True 使 key_filter 调用点无需再强制禁用 streamer 之外的后端。
- python/sglang/multimodal_gen/runtime/loader/weight_utils.py（模块 权重加载；类别 source；类型 dependency-wiring；符号 safetensors_weights_iterator）：加载入口接线改造，删除内联分支并把旧布尔参数映射为 reader 名字；任何回归都会影响全部

diffusion checkpoint 加载，是本 PR 行为保持的直接证据点。

- python/sglang/multimodal_gen/test/unit/test_weight_readers.py（模块 权重加载；类别 test；类型 test-coverage；符号 TestCapabilities, TestSelection）：10 个新单测固化能力标记与选择策略，覆盖显式请求、未知名报错、能力回退与环境变量决策，是本次重构行为保持的自动化保障。

关键符号：select_weight_reader, available_reader_names, WeightReader.is_available, WeightReader.iter_weights, RunaiStreamerReader.is_available, RunaiStreamerReader.iter_weights, SafetensorsMmapReader.is_available, SafetensorsMmapReader.iter_weights, safetensors_weights_iterator

关键源码片段

python/sglang/multimodal_gen/runtime/loader/weight_readers/base.py

定义 WeightReader 协议与两个能力标记，是整个重构的设计核心，把「能否跳过 key」「是否保留文件映射」从隐式规则变为显式契约。

```
# SPDX-License-Identifier: Apache-2.0
"""What a weight source has to provide, and what distinguishes one from another.

读取 checkpoint 曾经是一个布尔开关（Run:ai streamer 或 safe_open），但两条路径
的差异不止速度，还会决定正确性与内存行为，因此这里用显式能力标记来描述。
"""

from typing import Callable, ClassVar, Iterator, Protocol, runtime_checkable

import torch


@runtime_checkable
class WeightReader(Protocol):
    """按序产出（name, tensor）的权重读取器协议。

    supports_key_filter 表示能否在读取过程中跳过 key：Run:ai streamer
    会先物化全部 tensor 再逐个 yield，调用方即使只要一部分也省不了加载开销。

    retains_file_mapping 表示产出的 tensor 是否为 checkpoint 文件的视图：
    文件映射页在内存压力下可被内核直接回收（无需 swap），而 streamer 拷贝出的
    匿名页在无 swap 的主机上没有任何东西可回收。
    """

    name: ClassVar[str]
    supports_key_filter: ClassVar[bool]
    retains_file_mapping: ClassVar[bool]

    @classmethod
    def is_available(cls) -> bool:
        """返回该 reader 在当前安装环境中是否可用。"""
```

```
    def iter_weights(
        self,
        files: list[str],
        *,
        device: str,
        to_cpu: bool,
        key_filter: Callable[[str], bool] | None = None,
        clone_tensors: bool = True,
        show_progress: bool = True,
    ) -> Iterator[tuple[str, torch.Tensor]]:
        """按 reader 自有的顺序遍历 checkpoint 中的权重。"""
```

python/sglang/multimodal_gen/runtime/loader/weight_readers/__init__.py

select_weight_reader 集中了全部后端选择决策，含显式请求优先、fail-fast 报错与两级回退，是未来扩展新读取后端的唯一注册点。

```
# SPDX-License-Identifier: Apache-2.0
"""checkpoint 权重读取方式，以及选择其中一个的规则。"""

from sglang.multimodal_gen import envs
from sglang.multimodal_gen.runtime.loader.weight_readers.base import WeightReader
from sglang.multimodal_gen.runtime.loader.weight_readers.runai_streamer import
RunaiStreamerReader
from sglang.multimodal_gen.runtime.loader.weight_readers.safetensors_mmap import
SafetensorsMmapReader
from sglang.multimodal_gen.runtime.utils.logging_utils import init_logger

logger = init_logger(__name__)

# fallback 放最后且始终可用，保证任何选择路径都不会落空
_READERS: tuple[type, ...] = (RunaiStreamerReader, SafetensorsMmapReader)
FALLBACK_READER = SafetensorsMmapReader


def available_reader_names() -> list[str]:
    """返回当前环境可用的 reader 名字列表，供调用方做可发现性检查。"""
    return [b.name for b in _READERS if b.is_available()]


def select_weight_reader(*, requested: str | None = None, needs_key_filter: bool = False) ->
WeightReader:
    """选择 weight reader：显式请求优先，能力不满足时自动回退。

    requested 指定 reader 名字，None 表示由环境变量决定；未知名字直接抛
    ValueError（fail-fast），避免调用方拿到预期外的后端。选中后做两道兜底：
    未安装时回退到 fallback；调用方需要 key_filter 而选中项不支持时也回退，
    因为读完整份 checkpoint 再丢弃大部分比慢速读取所需部分更糟。
    """
    if requested is not None:
```

```
        chosen = next((b for b in _READERS if b.name == requested), None)
        if chosen is None:
            raise ValueError(
                f"unknown weight reader {requested!r}; "
                f"available: {available_reader_names()}"
            )
    elif envs.SGLANG_USE_RUNAI_MODEL_STREAMER and RunaiStreamerReader.is_available():
        chosen = RunaiStreamerReader
    else:
        chosen = FALLBACK_READER

    if not chosen.is_available():
        logger.info(
            "Weight reader %s is not installed; using %s",
            chosen.name,
            FALLBACK_READER.name,
        )
        chosen = FALLBACK_READER
    if needs_key_filter and not chosen.supports_key_filter:
        logger.debug(
            "Weight reader %s cannot skip keys at load time; using %s",
            chosen.name,
            FALLBACK_READER.name,
        )
        chosen = FALLBACK_READER
    return chosen()
```

python/sglang/multimodal_gen/runtime/loader/weight_readers/safetensors_mmap.py

safe_open 读取逻辑的封装，作为始终可用的 fallback；retains_file_mapping=True 是后续 host 内存回收方案的关键声明，且 supports_key_filter=True 使 key_filter 调用点无需再强制禁用 streamer 之外的后端。

```
# SPDX-License-Identifier: Apache-2.0
"""safe_open：读取较慢，但它是唯一能让页保持可回收的加载源。

safe_open 会 mmap checkpoint 文件，因此产出的 CPU tensor 是文件的视图
而非拷贝；这些页有文件背书，即使主机没有 swap，内核也能在内存压力下
直接丢弃它们。
"""

from typing import Callable, ClassVar, Iterator

import torch
from safetensors.torch import safe_open
from tqdm.auto import tqdm

_BAR_FORMAT = "{desc}: {percentage:.0f}%|{bar}| {n_fmt}/{total_fmt}"
```

```
class SafetensorsMmapReader:
    name: ClassVar[str] = "safetensors"
    # safe_open 按 key 逐个读取，天然支持在读取时跳过不需要的 key
    supports_key_filter: ClassVar[bool] = True
    # get_tensor 返回的 tensor 直接引用文件映射页，保留了可回收能力
    retains_file_mapping: ClassVar[bool] = True

    @classmethod
    def is_available(cls) -> bool:
        # 依赖 safetensors 库与 stdlib 的 mmap 能力，始终可用
        return True

    def iter_weights(
        self,
        files: list[str],
        *,
        device: str,
        to_cpu: bool,
        key_filter: Callable[[str], bool] | None = None,
        clone_tensors: bool = True,
        show_progress: bool = True,
    ) -> Iterator[tuple[str, torch.Tensor]]:
        # 注意 to_cpu 在本实现中没有直接使用，调用方通过把 device 设为
        # "cpu" 表达同一意图；clone_tensors 在此为 no-op，行为与重构前一致
        for path in tqdm(
            files,
            desc="Loading safetensors checkpoint shards",
            disable=not show_progress,
            bar_format=_BAR_FORMAT,
        ):
            with safe_open(path, framework="pt", device=device) as handle:
                for name in handle.keys():
                    if key_filter is not None and not key_filter(name):
                        continue
                    yield name, handle.get_tensor(name)
```

评论区精华

该 PR 没有实质性的 review 讨论（review 评论数为 0）。唯一的评论区动作是作者自己触发的 /tag-and-rerun-ci 命令，用于重跑 CI 门禁。合入质量主要依赖 PR body 中的论证与自动化验证：test/unit 从 1720 到 1730 全部通过（10 个新增为 reader 选择测试），作者还专门 revert 本改动复跑 4 个既有失败（realtime/、sana_wm/）以证明失败与本 PR 无关。

- CI 重跑请求 (other): CI 门禁重新运行；PR Test (Base) 通过，PR Test (Extra) 失败，AMD ROCm 7.2 仍在进行中。

风险与影响

- 风险：

1. 核心加载路径回归：weight_utils.py 的 safetensors_weights_iterator 是所有 diffusion 模型 checkpoint 加载的必经入口，虽然声明行为保持，但 mmap 路径从手工循环改为 yield from backend.iter_weights(...) 后，异常路径（损坏文件、重复 key）的语义差异未被端到端测试覆盖；且 test/unit 仍有 4 个既有失败（realtime/、sana_wm/）未解决，加载链路缺少全绿验证。
2. 外部符号导入位置变更：HAS_RUNAI_MODEL_STREAMER 从 weight_utils.py 顶层移到 weight_readers/runai_streamer.py，若仓库内其他模块从旧位置导入该符号会触发 ImportError；上下文材料未提供全仓库 grep 证据，存在不确定性。
3. 协议参数语义不对称：SafetensorsMmapReader.iter_weights 签名接受 clone_tensors 与 to_cpu，但实现中未消费这两个参数（to_cpu 语义由调用方通过 device="cpu" 表达，clone_tensors 为 no-op），当前行为与重构前一致，但未来调用方可能误以为所有实现都支持克隆语义。
4. 显式请求可能被静默替换：select_weight_reader 在 requested="runai_streamer" 且 needs_key_filter=True 时会回退到 mmap，日志级别为 debug，用户显式表达的后端偏好被覆盖时可能不易察觉。
5. 缺少文档更新：PR body checklist 中「Update documentation」未勾选，新协议与能力标记没有对外文档，其他模块接入时依赖代码注释。 - 影响：用户侧无运行时行为变化，加载速度与显存占用不变。系统侧，diffusion 加载链路获得了统一扩展点：后续新增读取后端（如 GGUF、远程存储流）只需实现协议并在 _READERS 注册；retains_file_mapping 与 supports_key_filter 能力标记为 host 内存规划（cgroup 预算、layerwise offload）提供了决策依据，直接衔接 #35641（pinned host memory 按 cgroup 规划）与 #35626（大 vocab 表驻留 host）的演进方向。团队侧，loader 维护者受益于集中选择逻辑与 fail-fast 报错，10 个新单测固化了选择契约。 - 风险标记：核心加载路径重构，外部符号导入位置变更，既有测试未全绿，接口契约存在未消费参数，缺少文档更新

关联脉络

- PR #35641 [diffusion] feat: plan pinned host memory against the cgroup cap not the machine: 同为 diffusion host 内存管理方向：本 PR 声明的 retains_file_mapping 能力标记，正是为这类 pinned host memory 预算规划提供「文件映射页可被内核回收」的决策依据。
- PR #35626 [diffusion] fix: keep large vocab tables in host memory under layerwise offload: layerwise offload 把大 vocab 表驻留 host 内存，与 retains_file_mapping 声明的内存语义直接相关；新协议可以服务这类 host 驻留决策。
- PR #35370 [diffusion] feat: load GGUF transformer checkpoints (MiniMax-H3): GGUF 是另一类 weight source 接入加载链路，改造方式与本 PR 一脉相承；新协议为这类后端提供了统一的注册与选择入口。
- PR #35418 [Diffusion] Support MiniMax-H3 pruned safetensors checkpoints: pruned safetensors 加载涉及 key 过滤与权重裁剪，与 supports_key_filter 能力以及 safetensors 加载路径直接相关。