PR #35688 完整报告

sgl-project/sglang

[diffusion] feat: let every layerwise component be configurable

合并时间：2026-08-20 22:38

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35688

执行摘要

- 一句话：layerwise 三旋钮支持按组件配置
- 推荐动作：值得精读，重点看 ServerArgs.layerwise_tuning_for 的三级默认设计（组件条目 > 组默认 > 辅助默认）与 configure_layerwise_offload 中从二值分支到统一取值的重构。PR body 对 residency 语义的澄清（静态划分而非缓存）是很好的设计论证范本，适合作为配置类功能的参考模式。

功能与动机

此前 configure_layerwise_offload 对 DiT 组之外的任何组件都直接替换为默认值（prefetch 0、resident 0、leading），而默认门的理由是辅助层每次组件使用只运行一次，“从 DiT residency 中得不到复用收益”。作者反驳了这一假设：residency 不是缓存而是静态分区，组件每请求使用一次也能每请求收回一次显存；prefetch 更是在单次 pass 内把传输与计算重叠。实测 H3 的 10.4 GB 视频 VAE 移出 layerwise 列表后 decode 从 39.85 s 降到 5.32 s（RTX 4090），而“移出列表”本质就是 resident = all，说明该轴对每请求一次的组件同样有效。因此正确做法是暴露旋钮而非替用户决定。

实现拆解

1. 参数定义（server_args.py）：在 ServerArgs 中新增三个字段 layerwise_prefetch_size、layerwise_resident_layers、layerwise_residency_policy，类型为 dict | str | None，支持与 --component-attention-backends 相同的 component=value 与 JSON 两种写法。
2. 解析与优先级（server_args.py）：新增静态方法 _parse_component_value_map 统一解析三种输入形态（dict、JSON 字符串、逗号分隔的 a=1,b=2）；新增 layerwise_tuning_for(component_name, dit_group=...) 返回 (prefetch, resident, policy) 三元组，优先级为 per-component 条目 > DiT 组默认（--dit-* 系列）> 辅助组件默认（0.0, 0.0, leading），并对非法 policy 抛出 ValueError。
3. 消费端改造（layerwise_offload.py）：configure_layerwise_offload 删除基于 dit_tuning_enabled 的二值分支，改为一次调用 layerwise_tuning_for 获取三值，再沿原有逻辑计算 prefetch_size、resident_layers 并将 residency_policy 直接传入 LayerwiseOffloadManager；pin_budget 与初始化顺序逻辑保持不变。
4. 测试与文档：test_layerwise_offload.py 新增 4 个用例（默认分组行为、per-component 命中不泄漏、未知 policy 报错、JSON 与 pair 两种拼写）；docs/docs/sglang-diffusion/api/cli.mdx 补充三个新选项的语义与权衡说明，并明确“未设

置条目时行为不变”。

关键文件：

- python/sglang/multimodal_gen/runtime/server_args/server_args.py（模块 参数配置；类别 source；类型 core-logic；符号 _parse_component_value_map, layerwise_tuning_for, _pick）：变更核心：新增三个 per-component 配置字段、_parse_component_value_map 解析器与 layerwise_tuning_for 优先级解析，是本次功能的入口与主逻辑。
- python/sglang/multimodal_gen/runtime/managers/memory_managers/layerwise_offload.py（模块 内存卸载；类别 source；类型 core-logic；符号 configure_layerwise_offload）：消费端主改动：configure_layerwise_offload 从 dit_tuning_enabled 二值分支改为统一调用 layerwise_tuning_for，是功能落地的关键路径。
- python/sglang/multimodal_gen/test/unit/test_layerwise_offload.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_layerwise_tuning_defaults_match_the_group, test_layerwise_tuning_per_component_entry_wins, test_layerwise_tuning_rejects_unknown_policy, test_layerwise_tuning_accepts_json_and_pair_forms）：新增 4 个单元测试覆盖默认分组行为、per-component 条目隔离、非法 policy 报错、JSON 与 pair 两种拼写，是默认行为不变的关键保障。
- docs/docs/sglang-diffusion/api/cli.mdx（模块 文档；类别 docs；类型 documentation）：补充三个 per-component 选项的用法、语义与权衡说明，明确默认行为不变，是功能可用的必要配套文档。

关键符号：_parse_component_value_map, layerwise_tuning_for, _pick, configure_layerwise_offload

关键源码片段

python/sglang/multimodal_gen/runtime/server_args/server_args.py

变更核心：新增三个 per-component 配置字段、_parse_component_value_map 解析器与 layerwise_tuning_for 优先级解析，是本次功能的入口与主逻辑。

```
@staticmethod
def _parse_component_value_map(
    value: dict[str, Any] | str | None, *, option: str
) -> dict[str, str]:
    # 与 --component-attention-backends 同一形态：接受 dict、JSON 或 a=1,b=2 字符串
    if value is None or value == "":
        return {}
    if isinstance(value, dict):
        return {str(k): str(v) for k, v in value.items()}
    if not isinstance(value, str):
        raise ValueError(
            f"{option} must be a dict or a comma-separated component=value string"
        )
    # 先按 JSON 解析，失败后再按逗号分隔的 component=value 解析
```

```
    try:
        parsed = json.loads(value)
        if isinstance(parsed, dict):
            return {str(k): str(v) for k, v in parsed.items()}
    except json.JSONDecodeError:
        pass
    result: dict[str, str] = {}
    for pair in value.split(","):
        pair = pair.strip()
        if not pair:
            continue
        if "=" not in pair:
            raise ValueError(f"{option} must use component=value entries")
        component, entry = pair.split("=", 1) # 只拆第一个等号，值里可含等号
        result[component.strip()] = entry.strip()
    return result


def layerwise_tuning_for(
    self, component_name: str | None, *, dit_group: bool
) -> tuple[float, float, str]:
    # 返回 (prefetch, resident, policy) 三元组：
    # 组件条目优先，否则 DiT 组回退到 --dit-* 标量，
    # 辅助组件保持旧的 (0.0, 0.0, leading)，因此默认行为不变
    prefetch_map = self._parse_component_value_map(
        self.layerwise_prefetch_size, option="--layerwise-prefetch-size"
    )
    resident_map = self._parse_component_value_map(
        self.layerwise_resident_layers, option="--layerwise-resident-layers"
    )
    policy_map = self._parse_component_value_map(
        self.layerwise_residency_policy, option="--layerwise-residency-policy"
    )

    def _pick(mapping: dict[str, str], group_default, aux_default):
        # 命中的组件条目优先；没有条目时按 DiT 组还是辅助组件决定默认值
        if component_name is not None and component_name in mapping:
            return mapping[component_name]
        return group_default if dit_group else aux_default

    prefetch = float(_pick(prefetch_map, self.dit_offload_prefetch_size, 0.0))
    resident = float(_pick(resident_map, self.dit_layerwise_resident_layers, 0.0))
    policy = str(
        _pick(
            policy_map,
            self.dit_layerwise_residency_policy,
            RESIDENCY_POLICY_LEADING,
        )
    )
```

```
    if policy not in RESIDENCY_POLICIES:
        # 非法 policy 在配置阶段立即暴露，而不是在流式执行时才出错
        raise ValueError(f"unknown residency policy {policy!r}")
    return prefetch, resident, policy
```

python/sglang/multimodal_gen/runtime/managers/memory_managers/layerwise_offload.py

消费端主改动：configure_layerwise_offload 从 dit_tuning_enabled 二值分支改为统一调用 layerwise_tuning_for，是功能落地的关键路径。

```
def configure_layerwise_offload(
    self,
    server_args: ServerArgs,
    *,
    pin_budget: HostPinBudget | None = None,
    component_name: str | None = None,
):
    self.layerwise_offload_managers = []
    named_modules = dict(self.named_modules())
    configured_layer_names = []
    # 原实现按 dit_tuning_enabled 二值分支，辅助组件直接取 (0.0, 0.0, leading)；
    # 现在统一交给 layerwise_tuning_for 解析，--dit-* 只是 DiT 组的默认值
    prefetch_value, resident_value, residency_policy = (
        server_args.layerwise_tuning_for(
            component_name,
            dit_group=self.layerwise_offload_dit_group_enabled,
        )
    )
    for layer_name in self.layer_names:
        module_list = named_modules.get(layer_name)
        if not isinstance(module_list, (torch.nn.ModuleList, torch.nn.Sequential)):
            continue
        if len(module_list) == 0:
            continue

        num_layers = len(module_list)
        # 小于 1.0 是层栈占比，>= 1 是绝对层数；MPS 上 0 表示不 prefetch
        if current_platform.is_mps() and prefetch_value == 0.0:
            prefetch_size = 0
        elif prefetch_value < 1.0:
            prefetch_size = 1 + int(round(prefetch_value * (num_layers - 1)))
        else:
            prefetch_size = int(prefetch_value)

        if resident_value <= 0:
            resident_layers = 0
        elif resident_value < 1.0:
            resident_layers = max(1, int(round(resident_value * num_layers)))
        else:
```

```
            resident_layers = min(num_layers, int(resident_value))

        # pinned 页内核无法回收，所以 pin_budget 不足时该组件放弃 pin，
        # prefetch 的异步拷贝就无法跑在计算前面
        pin_cpu_memory = server_args.pin_cpu_memory
        if pin_cpu_memory and pin_budget is not None:
            pin_cpu_memory = pin_budget.request(
                component_name=f"{component_name or type(self).__name__}.{layer_name}",
                weight_bytes=module_weight_bytes(module_list),
            )

        manager = LayerwiseOffloadManager(
            model=self,
            layers_attr_str=layer_name,
            num_layers=num_layers,
            enabled=True,
            pin_cpu_memory=pin_cpu_memory,
            prefetch_size=prefetch_size,
            resident_layers=resident_layers,
            initialize=False,
            residency_policy=residency_policy, # 不再由 dit_tuning_enabled 决定
        )
        self.layerwise_offload_managers.append(manager)
        configured_layer_names.append(layer_name)

    # 后续 MPS 立即 initialize，非 MPS 按参数量倒序 init 的逻辑保持不变
```

评论区精华

本 PR 没有外部 review 评论，唯一设计讨论来自作者在 PR body 中的论证：原门控将 residency 理解为“靠复用收益的缓存”，但 compute_streamed_layers 的文档语义是“哪些层被 streamed 而不是持留在 GPU 上”，--dit-layerwise-resident-layers 的 help 也说明 resident 层“启动时传输一次、每步不再重复传输”。因此 resident 是静态划分，每请求一次的组件同样能收回显存；prefetch 更是完全在单次 pass 内重叠传输与计算，与原门控理由无关。作者同时给出诚实的负面实测：text_encoder=2 时峰值显存从 22231 MiB 升到 24006 MiB（占卡 98%），但文本编码并没有变快，以此论证权衡是双向的，应该交由用户决策而不是替用户选值。

- 辅助组件是否应该拥有 resident/residency 旋钮 (design): 作者通过语义澄清与 MiniMax-H3 实测（VAE 移出列表 decode 39.85 s → 5.32 s）证实该轴对每请求一次的组件同样有效，决定暴露三个 per-component 旋钮，默认保持 (0.0, 0.0, leading) 以兼容旧行为。无外部 review 反对记录，合入者即作者本人。

风险与影响

- 风险：

1. 配置解析失败风险：layerwise_tuning_for 在每次配置时重新解析映射，非法 component=value 或未知 policy 会在启动阶段抛出 ValueError，可能阻断服务启动；好在错误信息指向明确选项名。
2. 显存超限风险：实测 text_encoder=2 时峰值达 24006/24564 MiB（98%），用户在不测量的情况下调大 prefetch 或 resident 可能触发 OOM，文档虽已提醒"先测量再调"，但仍属用户误配风险。
3. 默认兼容性：未设置新选项时行为与旧版完全一致（44 个既有单测通过），风险可控；但 layerwise_tuning_for 取代了原先内联的取值逻辑，属于配置核心路径改动，需要关注其他调用方是否有直接读取 dit_* 字段的习惯。
4. 缺少外部 review：review 评论为 0，仅作者自测与自审，合入质量依赖 CI（run-ci 通过）与测试覆盖，存在审查盲区。 - 影响：对用户：新增 3 个 CLI 选项，任何进入 layerwise-offload-components 的组件（含 text encoder、VAE）都能独立调优 prefetch、resident 与 residency policy，默认行为完全不变。对系统：把显存与延迟的权衡决策从代码硬编码移交给用户，且文档明确了每个旋钮的成本与适用场景。对团队：命令行 API 面扩展，后续新增组件无需改代码即可调优；文档与测试同步补齐，降低维护成本。影响范围限定在 diffusion 子系统的 layerwise offload 配置路径，不涉及 SRT 核心调度。 - 风险标记：配置解析失败会中断启动，显存峰值可达 98% 存在 OOM 风险，默认行为不变仅新增条目生效，无外部 review 记录

关联脉络

- PR #35641 [diffusion] feat: plan pinned host memory against the cgroup cap not the machine: 同一 layerwise_offload.py 文件与 offload 内存预算主线，pinned 内存按 cgroup 上限规划，与本 PR 的 prefetch/pin 逻辑直接相邻。
- PR #35626 [diffusion] fix: keep large vocab tables in host memory under layerwise offload: 同一 layerwise_offload.py 文件，处理 layerwise offload 下 host 驻留与显存峰值问题，与本 PR 的 resident 语义紧密相关。
- PR #35418 [Diffusion] Support MiniMax-H3 pruned safetensors checkpoints: 本 PR 的实测基于 MiniMax-H3，该 PR 提供了 H3 pruned checkpoint 加载能力，二者同属 diffusion 模型加载与内存管理演进线。