PR #25754 完整报告

sgl-project/sglang

[MLX] Support Qwen3.5 (dense) Model

合并时间：2026-05-30 17:05

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25754

执行摘要

- 一句话：支持 Qwen3.5 混合模型，重构 MLX 缓存体系
- 推荐动作：值得精读。特别是 attention_contract.py 的鸭式类型设计、auxiliary_state.py 的快照机制，以及混合批处理的性能优化思路。可作为 MLX 后端适配异构 Transformer 结构的参考。

功能与动机

Qwen3.5 hybrid models do not follow the older uniform self_attn decoder-layer layout. The MLX backend failed to start Qwen3.5 models because attention discovery assumed the first decoder layer was a standard softmax-attention layer, while Qwen3.5 can start with a linear-attention layer. Qwen3.5 also advertises image understanding, causing server warmup to select the VLM image path which fails on Apple Silicon MLX with 'Torch not compiled with CUDA enabled'.

实现拆解

1. 注意力鸭式类型检测(attention_contract.py)：新增 is_attention_module 等函数，通过运行时属性检测（如 q_proj, k_proj, rope, scale）确定模块是否为 softmax 注意力层，避免将 DeltaNet 等循环混合器误判。
2. 逐层注意力发现与打补丁(model_patching.py)：修改 find_attention_layers 和 _find_attention_attr，使其逐层扫描并返回每层注意力属性，支持 Qwen3.5 的异构布局。
3. 模型缓存布局(layout.py)：新增 MlxModelCacheLayout 数据类，将模型层映射到注意力池索引和辅助状态层索引，提供 first_attention_layer_index、has_auxiliary_state 等属性。
4. 辅助状态快照池(auxiliary_state.py)：新增 MlxAuxiliaryStatePool 类，为非 softmax 层（如 DeltaNet）提供原生 mlx-lm 缓存快照的保存与恢复，实现统一基数缓存的辅助状态组件。
5. 缓存类重命名与适配(attention_kv_cache.py)：将 ContiguousKVCache、PoolBackedCache、OffsetCache 分别重命名为 ContiguousAttentionKVCache、PoolBackedAttentionKVCache、AttentionOffsetCache，明确其注意力专用语义，并更新依赖路径。
6. 模型运行器改造(model_runner.py)：_acquire_cache 根据 MlxModelCacheLayout 分配注意力缓存和辅助状态缓存；新增 _new_native_cache 创建原生 mlx-lm 缓存对象；

_eval_with_cache 在 forward 前后管理辅助状态。

7. 混合批处理优化(继承自 alexnails 的提交) : 对辅助层进行能力检测，若原生缓存支持 merge/extract 则合并后批处理，否则逐请求串行，提升吞吐。
8. 预热路由调整(entrypoints/http_server.py) : 在 MLX 后端下，即使模型宣传图像理解能力，也强制走文本生成预热路径，避免因 torch.compile CUDA 问题崩溃。
9. 测试配套(test_attention_patching.py, test_unified_radix_cache_unittest.py) : 新增 1400+ 行单元测试，覆盖注意力打补丁、缓存布局辅助状态快照、布局感知、预热路由等场景。

关键文件:

- python/sglang/srt/hardware_backend/mlx/kv_cache/auxiliary_state.py（模块 辅助状态；类别 source；类型 dependency-wiring；符号 _clone_tree, _arrays_in_tree, collect, _CacheSnapshot）: 核心新文件：实现辅助状态快照池，支持统一基数缓存中非 softmax 层缓存的保存与恢复。
- python/sglang/srt/hardware_backend/mlx/model_runner.py（模块 模型运行；类别 source；类型 data-contract；符号 _acquire_cache, _new_cache_skeleton, _new_native_cache, _release_cache）: 核心变更：重构缓存分配、释放和 forward 流程，支持混合布局和辅助状态预取。
- python/sglang/srt/hardware_backend/mlx/kv_cache/layout.py（模块 缓存布局；类别 source；类型 dependency-wiring；符号 MlxModelCacheLayout, from_attention_discovery, num_layers, num_attention_layers）: 新增 MlxModelCacheLayout 将模型层映射到缓存池，支撑统一基数缓存的布局感知。
- python/sglang/srt/hardware_backend/mlx/kv_cache/attention_contract.py（模块 注意力契约；类别 source；类型 core-logic；符号 first_present_attr, get_num_heads, get_num_kv_heads, get_head_dim）: 核心新文件：提供鸭式类型注意力检测函数，为逐层打补丁和混合模型支持奠定基础。
- test/registered/unit/hardware_backend/mlx/test_attention_patching.py（模块 测试套件；类别 test；类型 test-coverage；符号 _set_runner_cache_layout, _set_runner_decode_context_defaults, _set_dummy_server_args_for_auxiliary_state_tests, TestMlxAttentionPatching）: 新增 1400+ 测试，覆盖注意力发现、缓存布局、辅助状态、预热路由等场景，保证稳定性。

关键符号: is_attention_module, first_present_attr, get_num_heads, get_num_kv_heads, get_head_dim, MlxModelCacheLayout.from_attention_discovery, _snapshot_cache, _restore_cache, MlxAuxiliaryStatePool.alloc, MlxAuxiliaryStatePool.free, find_attention_layers, _find_attention_attr, _acquire_cache, _new_native_cache, _eval_with_cache

关键源码片段

python/sglang/srt/hardware_backend/mlx/kv_cache/auxiliary_state.py

核心新文件: 实现辅助状态快照池，支持统一基数缓存中非 softmax 层缓存的保存与恢复。

```
# 辅助状态快照数据结构
```

```
@dataclass
class _CacheSnapshot:
    state: Any
    meta_state: Any = _MISSING
    attrs: dict[str, Any] | None = None

# 深度克隆 mx.array 树结构
def _clone_tree(value: Any) -> Any:
    if isinstance(value, mx.array):
        return mx.array(value)
    if isinstance(value, list):
        return [_clone_tree(item) for item in value]
    if isinstance(value, tuple):
        return tuple(_clone_tree(item) for item in value)
    if isinstance(value, dict):
        return {key: _clone_tree(item) for key, item in value.items()}
    return value

# 快照：克隆 cache 状态并强制求值
def _snapshot_cache(cache: Any) -> _CacheSnapshot:
    state = _clone_tree(getattr(cache, "state", ()))
    meta_state = (
        _clone_tree(cache.meta_state) if hasattr(cache, "meta_state") else _MISSING
    )
    attrs = {
        name: _clone_tree(getattr(cache, name))
        for name in _CACHE_ATTRS
        if hasattr(cache, name)
    }
    arrays = _arrays_in_tree((state, meta_state, attrs))
    if arrays:
        mx.eval(*arrays) # 保证所有数组已具体化
    return _CacheSnapshot(state=state, meta_state=meta_state, attrs=attrs)

# 恢复：将快照状态写回 cache 对象
def _restore_cache(cache: Any, snapshot: _CacheSnapshot) -> None:
    cache.state = _clone_tree(snapshot.state)
    if snapshot.meta_state is not _MISSING and hasattr(cache, "meta_state"):
        cache.meta_state = _clone_tree(snapshot.meta_state)
    for name, value in (snapshot.attrs or {}).items():
        setattr(cache, name, _clone_tree(value))

# 辅助状态池：管理多个请求的快照
def alloc(self, need_size: int) -> Optional[torch.Tensor]:
    if need_size > self.available_size():
        return None
    slots = self.free_slots[:need_size].clone()
    self.free_slots = self.free_slots[need_size:]
    for slot in slots.tolist():
```

```
            self._snapshots.pop(int(slot), None)
        return slots

    def free(self, indices: Any) -> None:
        if indices is None:
            return
        indices = self._tensor(indices)
        for slot in indices.tolist():
            self._snapshots.pop(int(slot), None)
        self.free_slots = torch.cat([self.free_slots, indices])
```

python/sglang/srt/hardware_backend/mlx/kv_cache/attention_contract.py

核心新文件：提供鸭式类型注意力检测函数，为逐层打补丁和混合模型支持奠定基础。

```
# 注意力模块必须包含的 attributes，包含 rope/scale 以防止循环混层误判
ATTENTION_API_ATTRS = ("q_proj", "k_proj", "v_proj", "o_proj", "rope", "scale")
NUM_HEAD_ATTRS = ("n_heads", "num_heads", "num_attention_heads")
NUM_KV_HEAD_ATTRS = ("n_kv_heads", "num_k_heads", "num_kv_heads", "num_key_value_
heads")

def is_attention_module(module: Any) -> bool:
    # 判断模块是否为标准的 softmax 注意力层
    return (
        all(hasattr(module, attr) for attr in ATTENTION_API_ATTRS)
        and get_num_heads(module) is not None
        and get_num_kv_heads(module) is not None
    )

def get_head_dim(module: Any) -> int | None:
    # 通过 head_dim 属性或 k_proj 权重推断 head_dim
    head_dim = first_present_attr(module, ("head_dim",))
    if head_dim is not None:
        return head_dim
    n_kv_heads = get_num_kv_heads(module)
    if n_kv_heads is None:
        return None
    if hasattr(module, "hidden_size") and hasattr(module, "num_k_heads"):
        return module.hidden_size // module.num_k_heads
    if hasattr(module, "k_proj") and hasattr(module.k_proj, "weight"):
        return module.k_proj.weight.shape[0] // n_kv_heads
    return None
```

评论区精华

- MlxAttentionKVPool 均匀性假设(jlee5814, category=design/performance): 质疑 MlxAttentionKVPool 假设每层共享相同的 n_kv_heads 和 head_dim，询问是负载约束还是简化。同时指出辅助状态快照在每一步都触发 mx.eval 同步，可能破坏重叠流水线。作者回应后提交了推迟快照的优化，并验证了性能改善。

- 鸭式类型检测命名(alexnails, category=style): 评论 duck taped 措辞，并询问是否可以从配置层获取信息。作者解释 MLX 需运行时检测，无法使用 CUDA 后端的 ModelConfig，并更新了文档字符串。
- 预热路由方案(yeahdongcn & alexnails, category=design): 讨论如何使用 SGLANG_USE_MLX 标志或类似方法，最终采用条件 not use_mlx() 屏蔽 VLM 路径，并标记 TODO 以支持未来图像服务。
- MlxAttentionKVPool 均匀性假设与辅助状态同步开销 (performance): 作者稍后提交了辅助状态快照的推迟优化（只在实际插入时执行），并测试验证了性能改善。均匀性假设在 v1 中可接受，未来可扩展为异构池。
- 鸭式类型检测的设计选择 (design): 作者解释 MLX 后端模型通过 mlx-lm 动态加载，无法使用 CUDA 后端的静态配置，必须运行时检测，因此鸭式类型是合适方案。随后更新了文档字符串澄清。
- VLM 预热路由的绕过方案 (correctness): 作者采用条件 not use_mlx() 实现绕过，并添加 TODO 注释。双方一致认为当前方案足够，未来 MLX 支持 VLM 时再启用。

风险与影响

- 风险:
  1. 向后兼容性: 缓存类（ContiguousAttentionKVCache 等）的重命名可能破坏外部对旧名的引用，但仅在仓库内使用，风险可控。
  2. 辅助状态同步开销: 早期版本在每一步都进行快照同步，虽已优化为只在引入时执行一次，但仍需关注 batch size 增大时的延迟是否恶化。
  3. 预热绕过 VLM: 强制文本生成会阻止 MLX 后端服务具备图像能力的模型，当前暂时正确，但未来需完善以支持 VLM。
  4. MLX 专属变更: 所有改动集中在 MLX 后端，不影响其他硬件后端，但引入了新的组件依赖（如 MlxAuxiliaryStateComponent、MlxModelCacheLayout），增加了维护工作。
- 影响:
  - 用户: MLX 后端（Apple Silicon）用户现在可以运行 Qwen3.5 密集模型与混合模型（如 Qwen3.5-0.8B），并获得基数缓存加速。非 MLX 用户无影响。
  - 系统: 缓存架构重构使 MLX 支持混合注意力布局，统一基数缓存的使用使预填充延迟降低 50%-78%。
  - 团队: MLX 模块代码量增加约 3000 行，引入了新的设计模式（鸭式类型、辅助状态快照池），需要团队掌握。
  - 风险标记: 核心路径变更，MLX 专属重构，辅助状态同步开销，VLM 预热绕过，类名重命名破坏兼容

关联脉络

- PR #26549 [UnifiedTree]: Support eviction priority: 本 PR 依赖统一基数缓存组件（如 TreeComponent、MambaComponent），该 PR 对其进行了扩展（驱逐优先级），两者共同构建了混合模型的缓存基础。