PR #30050 完整报告

sgl-project/sglang

[MLX] Support gpt-oss: sliding-window attention, attention sinks, sm_scale

合并时间：2026-08-10 09:32

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30050

执行摘要

- 一句话：MLX 后端支持 gpt-oss：滑动窗口注意力、attention sinks、sm_scale
- 推荐动作：值得精读。关注三个设计决策：一是“读取时应用窗口 vs 旋转缓存”的取舍——保留完整 KV 并用 banded mask/ 尾部截断，在 softmax 下数值等价且不破坏 radix 前缀复用，代价是显存冗余，这是很实用的权衡；二是测试防假阳性设计——刻意要求提示词 >128 token 保证窗口真正参与，否则短提示词下窗口层和全注意力输出相同、测试会空过；三是 AOT 内核门控的 fail-safe 策略——宁可回退到 mx.fast.rope 也绝不用 vanilla 内核算错 scaled RoPE。做平台后端或注意力系统的人可参考其中的契约设计与测试方法论。

功能与动机

PR body 明确指出 gpt-oss 无法在 MLX 后端服务：mlx_lm 的 gpt_oss AttentionBlock 把 softmax scale 命名为 sm_scale 而契约只接受 scale，导致 find_attention_layers 加载即抛 ValueError；MLX 后端完全没有滑动窗口支持，window=128 的层在 prefill/extend 期间静默退化为全注意力（短提示词测试无法暴露）；batched decode 不应用窗口、丢弃 sinks 并直接读 inner.scale；可选的 AOT Metal RoPE 内核会错误接受 YarnRoPE 并算出 base=10000 的 vanilla RoPE；此外 server_args.py 的 GptOss 分支强制 attention_backend=triton，在 macOS 上导致启动即崩溃。

实现拆解

按四层拆解实现：

1. 契约层（attention_contract.py）：把 scale 从必选属性改为 SCALE_ATTRS = ("scale", "sm_scale") 二选一，新增 get_attention_scale()；新增 WINDOW_SIZE_ATTRS = ("window_size", "sliding_window") 兼容 gpt_oss/gemma4 与 olmo3/llama 的命名差异，get_container_window_size() 读取容器级标量窗口，get_layer_window_sizes() 依据 mlx-lm 的 layer_types 惯例把窗口映射到每个具体层（"sliding_attention" 标记窗口层，返回 {layer_idx: window|None}）。is_attention_module() 同步放宽为“任一 scale 属性存在”，避免误拒 gpt_oss 同时继续排除 DeltaNet 等非注意力模块。
2. Mask 层（attention_kv_cache.py）：新增模块级 make_attention_mask(N, offset, return_array, window_size)，直接委托 mlx_lm 自身的 create_causal_mask 生成 banded mask，关键点是 window_size 非 None 时 N==1 也返回真实 mask（此前返回 None 会导致 decode 阶段窗口被静默禁用）；三个 cache shim（AttentionOffsetCache、ContiguousAttentionKVCache、PoolBackedAttentionKVCache）的 make_mask 全部改

为委托该函数，修复滑窗层 prefill 退化为全注意力的问题。

3. 解码层（attention_wrapper.py + model_patching.py）：MLXAttentionWrapper.__init__ 新增可选 window_size，并在加载 / 打补丁时一次性解析并缓存 _scale 与 _sinks（热路径不再扫描属性，scale 缺失则 fail-fast）；_batched_decode 对窗口层按 min(seq_len+1, window) 截断每个请求 KV 的尾部、按窗口长度重建 padding mask（上下文里的 padding 元数据是全量长度的）、仅当模块有 sinks 时才把 sinks= 传给 mx.fast.scaled_dot_product_attention。patch_model_attention 把 get_layer_window_sizes 的结果逐个传给 wrapper，并在“声明了 scalar 窗口但没有 layer_types 映射”时（如 gemma3 风格模型）打 WARNING，避免 decode 与 prefill 语义分裂而无人知晓。
4. 平台接线与安全回退（model_runner_stub.py / server_args.py / arg_groups/overrides.py / aot.py）：MlxModelRunnerStub 覆盖 init_attention_backends() 为 no-op（attn_backend = None），防止基类按 server_args 命名构造真实后端时在 _DummyKVCache 上崩溃；server_args.py 的 gpt-oss 支持后端断言与 arg_groups/overrides.py 的后端强制均改为 elif not is_mps() 平台谓词，MLX 服务时保留平台默认 torch_native，非 Apple Silicon 路径行为完全不变；aot.py 的 _build_rope_kernel 在识别到 base 缺失、预计算 _freqs、mscale != 1 或线性 scale != 1 时一律回退到 mx.fast.rope，避免用 vanilla 内核算错 scaled RoPE。
5. 测试与 CI 配套：新增 test_sliding_window_attention.py（24 个单测：契约接受、窗口解析两个命名、mask 与 mlx_lm 参考的网格比对、banded 语义、N==1 仍保持、batched decode 与手工参考 float-tight 对比、sinks 语义、AOT 门控）与 test_gpt_oss_mlx_correctness.py（黑盒 serving smoke + MlxModelRunner 与未打补丁 mlx_lm 的逐 token 参考等价，提示词强制 >128 token 以真正触发窗口）；按 #30121 的 suite 注册机制挂到 base-a-test-cpu、stage-a-unit-test-mlx、stage-b-e2e-mlx；test_mlx_runner_pool_contract.py 增加对 init_attention_backends 覆盖的防漂移断言。

关键文件：

- 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 get_attention_scale, get_container_window_size, get_layer_window_sizes）：注意力契约的核心改动：scale 属性放宽为 scale/sm_scale 二选一，新增 per-layer 窗口解析（layer_types + window_size/sliding_window），是所有 MLX 注意力模型加载与识别的入口。
- python/sglang/srt/hardware_backend/mlx/kv_cache/attention_kv_cache.py（模块 KV 缓存层；类别 source；类型 core-logic；符号 make_attention_mask, make_mask）：新增 make_attention_mask 统一三个 cache shim 的 make_mask，修复滑动窗口层 prefill/extend 静默退化为全注意力的问题；N==1 也保持 banded mask 是 decode 正确性的关键。
- python/sglang/srt/hardware_backend/mlx/kv_cache/attention_wrapper.py（模块 解码包装器；类别 source；类型 core-logic；符号 init, _batched_decode）：MLXAttentionWrapper 是 batched decode 的核心执行单元：新增 window_size 支持、scale/sinks 缓存、窗口截断 + 局部 padding mask，无窗口模型保持原路径。

- python/sglang/srt/hardware_backend/mlx/kv_cache/model_patching.py（模块 模型打补丁；类别 source；类型 data-contract；符号 patch_model_attention）：把 per-layer window sizes 接入 MLXAttentionWrapper，并针对"有窗口但无 layer_types 映射"的模型告警，防止 decode 与 prefill 语义分裂。
- python/sglang/srt/hardware_backend/mlx/aot.py（模块 AOT 内核；类别 source；类型 core-logic；符号 _build_rope_kernel）：AOT Metal RoPE 内核门控收紧：拒绝 YarnRoPE 等 scaled 变体回退 mx.fast.rope，避免 gpt-oss 开启自定义内核时算错旋转角。
- python/sglang/srt/hardware_backend/mlx/model_runner_stub.py（模块 运行器桩；类别 source；类型 data-contract；符号 init_attention_backends）：覆盖 init_attention_backends 为 no-op 并保持 attn_backend = None，防止 server_args 命名的 torch 后端在 MLX stub 的 _DummyKVCache 上构造崩溃。
- python/sglang/srt/server_args.py（模块 服务端参数；类别 source；类型 core-logic）：gpt-oss 的 CUDA 后端断言在 MPS 上会因 attention_backend 未设置而误失败，需用平台谓词收窄；同时配套 arg_groups/overrides.py 的后端强制逻辑。
- test/registered/unit/hardware_backend/mlx/test_sliding_window_attention.py（模块 滑动窗口测试；类别 test；类型 test-coverage；符号 _tiny_gpt_oss_model, TestGptOssAttentionContract, test_gpt_oss_attention_passes_contract, test_get_attention_scale_prefers_scale_over_sm_scale）：24 个单元测试覆盖契约、mask 与 mlx_lm 参考网格比对、batched decode 与手工参考严格对比、AOT 门控，是本次功能正确性的主要防线。
- test/registered/mlx/models_e2e/test_gpt_oss_mlx_correctness.py（模块 端到端测试；类别 test；类型 test-coverage；符号 _available_gb, TestGptOssMlxCorrectness, test_basic_generation_nonempty, test_simple_arithmetic）：端到端正确性验证：黑盒 serving smoke + 与未打补丁 mlx_lm 的逐 token 参考等价，提示词强制 >128 token 确保窗口真正参与，是防假阳性的关键设计。
- test/registered/unit/hardware_backend/mlx/test_mlx_runner_pool_contract.py（模块 桩契约测试；类别 test；类型 test-coverage；符号 test_stub_overrides_base_init_attention_backends, test_stub_init_attention_backends_keeps_attn_backend_none）：防漂移守护：断言 MlxModelRunnerStub 仍覆盖 init_attention_backends 且 attn_backend 保持 None，防止未来重构丢失该覆盖导致 MLX 启动崩溃。

关键符号：get_attention_scale, get_container_window_size, get_layer_window_sizes, make_attention_mask, MLXAttentionWrapper.init, MLXAttentionWrapper._batched_decode, patch_model_attention, _build_rope_kernel, MlxModelRunnerStub.init_attention_backends

关键源码片段

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

注意力契约的核心改动：scale 属性放宽为 scale/sm_scale 二选一，新增 per-layer 窗口解析（layer_types+window_size/sliding_window），是所有MLX注意力模型加载与识别的入口。

```
# 注意力契约：mlx-lm 各模型对 softmax scale 的命名不统一，
# gpt_oss 使用 sm_scale，而大多数模型使用 scale。
```

```
SCALE_ATTRS = ("scale", "sm_scale")

# mlx-lm 容器对滑动窗口标量的命名有两种：window_size（gpt_oss、gemma4）
# 与 sliding_window（olmo3、llama SWA 变体）。
WINDOW_SIZE_ATTRS = ("window_size", "sliding_window")


def get_attention_scale(module: Any) -> float | None:
    # 任选其一即可满足契约；first_present_attr 按元组顺序取第一个存在属性。
    return first_present_attr(module, SCALE_ATTRS)


def get_container_window_size(model: Any) -> int | None:
    # 逐层定位容器：language_model -> model，读取容器级 scalar 窗口。
    root = getattr(model, "language_model", model)
    container = getattr(root, "model", root)
    return first_present_attr(container, WINDOW_SIZE_ATTRS)


def get_layer_window_sizes(model: Any) -> dict[int, int | None]:
    """按 mlx-lm 容器惯例解析每层窗口大小。

    gpt_oss / olmo3 等容器暴露 layer_types（每层一个条目，
    "sliding_attention" 标记窗口层）配一个 scalar 窗口。
    返回 {layer_idx: window 或 None}；不遵循该惯例时返回空 dict。
    """
    root = getattr(model, "language_model", model)
    container = getattr(root, "model", root)
    layer_types = getattr(container, "layer_types", None)
    window_size = get_container_window_size(model)
    if not layer_types or window_size is None:
        return {}
    return {
        idx: window_size if layer_type == "sliding_attention" else None
        for idx, layer_type in enumerate(layer_types)
    }
```

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

MLXAttentionWrapper 是 batched decode 的核心执行单元：新增 window_size 支持、scale/sinks 缓存、窗口截断 + 局部 padding mask，无窗口模型保持原路径。

```
# 滑动窗口层：pool 保留完整 KV 历史，但最新 token 只关注尾部 window 个 key。
# 对 softmax 注意力而言，这与旋转缓存数值等价，同时不破坏 radix 前缀复用。
window = self._window_size
if window is None:
    # 无窗口模型走原路径，pad 元数据直接复用上下文里的全量版本。
    pad_sizes = ctx.pad_sizes
else:
    # 上下文共享的 padding 元数据按全量长度计算，
```

```
        # 窗口层必须按 min(seq + 1, window) 重建有效长度与 pad。
        eff_lens = [min(n + 1, window) for n in ctx.seq_lens]
        max_eff = max(eff_lens)
        pad_sizes = [max_eff - n for n in eff_lens]

    for i in range(B):
        layer_caches[i].write_token(keys[i : i + 1], values[i : i + 1])
        k_all, v_all = layer_caches[i].get_kv()
        if window is not None and k_all.shape[2] > window:
            # 只保留尾部 window 个 key/value，效果等价于 banded mask。
            k_all = k_all[:, :, -window:, :]
            v_all = v_all[:, :, -window:, :]

        pad = pad_sizes[i]
        if pad > 0:
            # 补齐对齐 batch 长度，pad 位置后续会被 mask 屏蔽。
            k_pad = mx.zeros((1, n_kv_heads, pad, head_dim), dtype=k_all.dtype)
            v_pad = mx.zeros((1, n_kv_heads, pad, head_dim), dtype=v_all.dtype)
            k_all = mx.concatenate([k_all, k_pad], axis=2)
            v_all = mx.concatenate([v_all, v_pad], axis=2)
```

评论区精华

Review 中几个关键交锋：

- server_args 平台特殊化方向：yeahdongcn 反对在 server_args.py 引入 _mlx_serving = is_mps() and use_mlx() 特例，认为让通用验证路径理解 MLX 细节是错误方向。作者回应：纯 revert 会破坏启动，因为 MPS 上 attention_backend 在断言执行前仍是 None（torch_native 默认值稍后才填充）；最终采用 elif not is_mps() 平台谓词同时收束 overrides 与断言，既保留非 MPS 的快速失败，又不污染 MLX 路径。
- 热路径缓存与 fail-fast：gemini-code-assist 建议把 get_attention_scale 与 getattr(inner, "sinks", None) 从每次 decode 的热路径移到 __init__ 一次性解析，既省掉每次属性扫描，又能在加载期尽早暴露坏模块。作者采纳，_scale 与 _sinks 均在 patch 时缓存。
- AOT RoPE 门控需复核：yeahdongcn 对 _build_rope_kernel 拒绝 scaled RoPE 的判定（_freqs、mscale、scale）@adityavaid 复核，后续无异议。
- 测试专用 env 注册：yeahdongcn 建议 SGLANG_MLX_TEST_* 只在测试文件里读 os.environ，不要进入全局 Envs 注册表；作者同意并移除。
- 注释风格：yeahdongcn 指出 model_runner_stub.py 注释“读起来像 AI 生成的”要求精简，作者已修剪。
  - server_args.py 平台特殊化的设计方向 (design): 作者说明纯 revert 会导致 MPS 上 attention_backend 为 None 时断言先行失败，最终保留断言并用 if not is_mps() 平台谓词同时收束 overrides 与 server_args。
  - attention scale 与 sinks 的热路径缓存 (performance): 作者采纳：_scale 与 _sinks 在 wrapper 构造时解析并缓存，scale 缺失直接抛 RuntimeError。

- AOT RoPE 门控的正确性复核 (correctness): 复核后无异议，门控逻辑随 PR 合入；默认路径不受影响（SGLANG_MLX_USE_CUSTOM_ROPE 默认关闭）。
- 测试专用环境变量是否进入全局注册表 (design): 作者同意："Sure, it makes sense"，注册被移除，e2e 测试直接读 os.environ。
- stub 注释风格 (style): 作者回复 "Done, trimmed"，注释已精简。

风险与影响

- 风险：
  - 窗口层 KV 冗余存储：设计上窗口层在 pool 里保留完整 KV 历史、读取时截断，长序列下会持续占用不被再读的 KV slot，PR 自述将 per-layer windowed pool 留作 TODO。极端长连接 + 多层滑窗会放大内存占用。
  - gemma3 风格模型语义分裂：model_patching.py 只对"没有 layer_types 但有 scalar window"的模型告警，decode 实际不应用窗口；若未来接入该类模型，>128 token 的输出会静默错误，目前仅靠日志提示。
  - AOT 门控依赖 mlx 内部属性：_build_rope_kernel 检查 _freqs、mscale、scale 等非公共属性，mlx 升级可能改变命名或语义，导致门控错误放行（算错）或误拒绝（性能回退）；需要依赖版本测试守护。
  - 注意力契约放宽的波及面：ATTENTION_API_ATTRS 把 scale 移出必选集合，is_attention_module 改为任一 scale 属性，所有 MLX 模型加载与 find_attention_layers 都会经过新判定；若未来某模型把 sm_scale 用于非 softmax 语义，可能被误判为注意力模块。
  - CI 覆盖缺口：新增测试全部依赖 mlx/mlx_lm，当前 CI runner 全部跳过，真实回归只能依赖 Apple Silicon 本机或门控的 macOS lane，存在覆盖盲区。
  - 影响：对 Apple Silicon 用户，gpt-oss 系列（如 gpt-oss-20b-MXFP4-Q8）现在可直接通过 SGLANG_USE_MLX=1 服务，且贪心解码与未打补丁 mlx_lm 逐 token 一致；已用 >128 token 提示词验证窗口真正参与计算。对既有 MLX 模型，无窗口 decode 路径逻辑保持字节级不变，qwen MoE 参考等价与服务 smoke 均通过。对非 MLX 平台（CUDA/ROCm/XPU/CPU），改动仅用 if not is_mps() 收窄了 gpt-oss 后端断言与 override，行为完全不变。对团队而言，这是 Apple 支持 roadmap（#19137）的关键里程碑，为后续 olmo3、gemma4、llama SWA 变体等窗口模型铺平了契约与基础设施。
  - 风险标记：窗口层 KV 全量存储未裁剪，gemma3 风格模型 decode 不应用窗口（仅告警），AOT RoPE 门控依赖 mlx 内部属性，非 Apple Silicon CI 全部跳过，注意力契约放宽影响所有 MLX 模型加载

关联脉络

- PR #19137 [Roadmap] Apple Device Support (2026 Q2): 本 PR 直接承接 roadmap 的 gpt_oss 条目，PR body 开头即标注 Part of #19137，是 Apple Silicon 支持路线图上的关键功能补齐。
- PR #29440 qwen MoE MLX correctness tests structure: PR body 明确说明正确性测试结构遵循 qwen MoE MLX 测试（#29440），包括 serving smoke 与参考等价的组织方式。

- PR #30121 MLX CI suite registration mechanism: commit 说明滑动窗口单测与 gpt-oss e2e 测试按 #30121 的 run_suite.py suite 注册机制挂到 stage-a-unit-test-mlx / stage-b-e2e-mlx。
- PR #29217 forward_ct accounting change: commit 2f6407b 对齐 FakeOverlapScheduler stub 与 #29217 的 forward_ct 记账方式，属于本 PR 处理 main 分支冲突时的配套调整。