PR #34166 完整报告

sgl-project/sglang

[MLX] Window-bounded SWA KV storage and in-graph sampling

合并时间：2026-08-10 12:28

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34166

执行摘要

- 一句话：MLX 后端窗口化 SWA KV 存储并新增图内采样
- 推荐动作：值得精读。这是 MLX 后端迄今最重要的 PR 之一，建议重点关注三个设计决策：
  1. 图内 Gumbel-max 采样如何与 overlap 调度器共存——chaining 属性保住了 0.05-1.2 ms/step，而 body 诚实披露采样本身在 batch 64 时要 0.5-24 ms/step，这个取舍框架可以直接借鉴；
  2. SWA 前缀全量重算的取舍——用精确性论证代替启发式优化，并明确标注 " 后续 SWARadixCache 恢复快速命中 " 的演进路径；
  3. bounded top-K 链与 murmur hash 移植——把 CUDA 语义搬到 mx 平台时如何保持 seeded 行与 batch 组成无关的契约。 同时注意其性能数字都是低并发 Apple Silicon 本地 serving 场景，推广到高并发服务前需重新验证。

功能与动机

PR body 明确指出 MLX 后端的两个缺口阻塞 gpt-oss 类模型："Sliding-window layers stored full KV history"（#30050 只做了读时加窗，滑动层仍按全长上下文付费）；"Token selection was unconditionally greedy"（temperature、top_p、top_k、min_p 和 seeded sampling 对 MLX 无效）。此外，body 用大量实测数据论证了为何采样必须图内化而非桥接 CPU torch Sampler（#25804 方案在默认 overlap 调度路径上不可达，且 CPU-side sort 是真正瓶颈），以及为何 SWA 前缀命中必须全量重算而非重建 trailing band（感受野链式依赖导致不精确）。

实现拆解

整个变更分六步落地：

1. 窗口有界 SWA KV 存储：在 python/sglang/srt/hardware_backend/mlx/kv_cache/attention_kv_cache.py 新增 WindowedAttentionKVCache，固定容量 window + max(chunk, window) 的时序缓冲，decode 原地追加、满时压缩（摊还 O(1)），offset 保持绝对坐标使 RoPE 位置与调度簿记不变；make_mask 将 mask offset 钳制到保留前缀，保证 mask 宽度恒等于返回 key 长度。同时优化 make_attention_mask：offset + N <= window_size 时窗口无法绑定，直接退回普通 causal（避免物化 banded mask 把 SDPA 赶出融合 causal 路径，约 2 倍每层开销）。BatchedDecodeContext.decode_padding 按 window 缓存 pad 尺寸与 keep mask，gpt-oss 每步 24 个注意力层只需构建两次。

2. radix 共享池按层类型拆分：python/sglang/srt/hardware_backend/mlx/kv_cache/layout.py 的 MlxModelCacheLayout 新增 layer_window_sizes、full_attention_layer_indices/swa_attention_layer_indices 分区与稠密 full_kv_pool_index_by_layer；MlxAttentionKVPool 只存 full-attention 层；SWA 模型 init_cache_pools 直接提前返回（实测 gpt-oss-20b 省回 6.58 GiB 且池子本就无读者）。SWA 前缀命中改为全量重算：body 用实验证明 trailing-band 重建会让 full-attention K 偏移高达 2.1 且贪心输出发散，因此接受 " 命中无 prefill 加速 " 的取舍（hit/cold 耗时比 1.00-1.01）。
3. 图内采样核心：新增 python/sglang/srt/hardware_backend/mlx/sampling.py（491 行）：MlxSamplingParams.from_req 在 prefill 注册时冻结参数并处理 seed 契约；sample_tokens 用 Gumbel-max 恒等式替代 multinomial，纯 mx 算子留在懒图内；MAX_BOUNDED_TOP_K = 1024 时启用 [B, K] 候选链而非全词表链；seeded 行移植 CUDA MurmurHash3(seed, position, token_id) 到 mx uint32 运算；u = 1 clamp 与 #33423 在 CUDA 路径修复的 +inf 风险独立达成一致。
4. runner 与 worker 接线：model_runner.py 新增 _select_tokens_with_logprobs、_edited_logits，MlxPendingPrefill/Extend/Decode 扩展 lazy_logprobs、logprob_spec、edit_rows 以支撑 chained 步；tp_worker.py 用 MlxLaunch dataclass 取代 5 元组 launch 协议，新增 _build_logit_edit_rows（预组合 grammar mask + logit_bias 为加法行）、_logprob_spec_for、_custom_logits_hook；同步路径 _forward_batch_generation_mlx 重写为 async path + finalize 背靠背，删除重复实现。
5. 调度约束 chain_safe：scheduler_mixin.py 新增 _mlx_batch_chain_safe——grammar/custom logit processor 批次需要上一步 token 物化才能构建下一步 mask，因此禁止 chaining，每步 fresh launch；_launch_chained 改用 dataclasses.replace 保留字段。
6. 公共平台修复与文档：xgrammar_backend.py 的 _allocate_token_bitmask 的 pin_memory=True 改为平台感知、apply_vocab_mask 增加 cpu 分支（无 CUDA 主机的通用修复）；arg_groups/overrides.py、server_args.py 将 gpt-oss 的 MPS carve-out 收紧为 is_mps() and use_mlx()；配套 CLI flag、文档与 6 个测试文件（覆盖采样、窗口缓存等价性、SWA radix/pool 契约、tp_worker 路由、滑动窗口 attention、gpt-oss e2e）。

关键文件：

- python/sglang/srt/hardware_backend/mlx/sampling.py（模块 采样器；类别 source；类型 core-logic；符号 MlxSamplingParams, from_req, is_greedy, MlxLogprobSpec）：新增的图内采样核心模块（491 行），用纯 mx 算子实现 temperature/top-k/top-p/min-p/seed 采样、Gumbel-max 替代 multinomial、bounded top-K 优化与 murmur hash 移植，是 PR 新功能的主载体。
- python/sglang/srt/hardware_backend/mlx/model_runner.py（模块 模型运行器；类别 source；类型 data-contract；符号 _eval_with_cache, _cache_state_arrays, cache_state_arrays, decode_batch）：MLX 运行器主路径接线：接入 WindowedAttentionKVCache、采样模块、_select_tokens_with_logprobs、pending 结构扩展，是窗口缓存与采样两大特性的汇合点。

- python/sglang/srt/hardware_backend/mlx/tp_worker.py（模块 工作进程；类别 source；类型 dependency-wiring；符号 MlxLaunch, _forward_batch_generation_mlx, _chunk_needs_logits, _sampling_active）：Worker 层依赖接线：MlxLaunch dataclass 取代 5 元组协议，新增 logit 编辑行构建、logprob 规格与自定义 logit hook，同步路径重写为 async + finalize 背靠背。
- python/sglang/srt/hardware_backend/mlx/kv_cache/attention_kv_cache.py（模块 窗口 KV 缓存；类别 source；类型 core-logic；符号 get_kv, reset, WindowedAttentionKVCache, init）：新增 WindowedAttentionKVCache（窗口有界 SWA KV 存储）并优化 make_attention_mask 的窗口无法绑定短路，是 Part 1 的核心实现。
- python/sglang/srt/hardware_backend/mlx/scheduler_mixin.py（模块 调度器；类别 source；类型 core-logic；符号 _mlx_batch_chain_safe）：引入 chain_safe 调度约束：grammar/custom logit processor 批次禁止 chaining，是采样与 overlap 调度器正确共存的机制保障。
- python/sglang/srt/hardware_backend/mlx/kv_cache/attention_wrapper.py（模块 注意力封装；类别 source；类型 core-logic；符号 decode_padding）：batched decode 的 padding 按 window 缓存构建、头部形状元数据缓存化，并补齐 full_kv_pool_index 守卫，直接支撑窗口化 decode 性能。
- 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 post_init, num_full_attention_layers, has_sliding_window_layers, window_size）：MlxModelCacheLayout 完成 full/SWA 层分区与稠密 full_kv_pool_index，是共享池拆分的结构基础，也是后续 window-aware SWA 池的接缝。
- test/registered/unit/hardware_backend/mlx/test_mlx_sampling.py（模块 采样器；类别 test；类型 test-coverage；符号 _reference_murmur3, mix, _params, TestMurmurHashPort）：728 行采样单测：murmur hash 与纯 Python 参考逐位对齐、seeded 行与 batch 组成无关、bounded top-K 与全词表链同 token、u=1 clamp 防护验证。
- test/registered/unit/hardware_backend/mlx/test_windowed_kv_cache.py（模块 窗口缓存测试；类别 test；类型 test-coverage；符号 _dense_mask, _tiny_gpt_oss_model, TestWindowedCacheEquivalence, _kv）：窗口缓存与全历史 trailing slice 的三层级等价性测试（缓存数组、容器 forward、batched decode），是窗口存储正确性的核心保障。
- test/registered/unit/hardware_backend/mlx/test_swa_radix_pool.py（模块 SWA 池测试；类别 test；类型 test-coverage；符号 _tiny_gpt_oss_model, _stub_runner, TestSwaLayoutAndPoolContract, _layout）：SWA 模型共享池契约测试：层分区、SWA 模型无池、all-SWA 无池、sync 仅写 full 层，为后续 window-aware SWA 池保留接缝验证。

关键符号：MlxSamplingParams.from_req, sample_tokens, compute_logprobs, scale_by_temperature, WindowedAttentionKVCache._append, WindowedAttentionKVCache.update_and_fetch, MlxModelCacheLayout.post_init, MlxModelCacheLayout.full_kv_pool_index, BatchedDecodeContext.decode_padding, MLXAttentionWrapper._batched_decode, SchedulerMlxOverlapMixin._mlx_batch_chain_s

afe, MlxTpModelWorker._build_logit_edit_rows, MlxTpModelWorker._forward_batch_generation_mlx, MlxModelRunner._select_tokens_with_logprobs, make_attention_mask

关键源码片段

python/sglang/srt/hardware_backend/mlx/sampling.py

新增的图内采样核心模块（491 行），用纯 mx 算子实现 temperature/top-k/top-p/min-p/seed 采样、Gumbel-max 替代 multinomial、bounded top-K 优化与 murmur hash 移植，是 PR 新功能的主载体。

```
# sampling.py 片段 1：参数冻结契约
@classmethod
def from_req(cls, req: Any, deterministic_seeding: bool = False) -> MlxSamplingParams:
    sp = req.sampling_params
    # 惩罚项（frequency/presence/repetition）依赖 Triton kernel，
    # Metal 上不可用；每个进程只告警一次，避免刷屏。
    global _warned_ignored_penalties
    if not _warned_ignored_penalties and (
        sp.frequency_penalty != 0.0
        or sp.presence_penalty != 0.0
        or sp.repetition_penalty != 1.0
    ):
        _warned_ignored_penalties = True
        logger.warning(
            "MLX sampling ignores frequency/presence/repetition penalties; "
            "a request specified them. (Warning logged once.)"
        )
    # seed 契约与其他后端完全一致：只有
    # --enable-deterministic-inference 开启时 sampling_seed 才生效，
    # 且此时所有行都会被播种（未指定 seed 的行用 42）。
    seed = None
    if deterministic_seeding:
        seed = (
            sp.sampling_seed
            if sp.sampling_seed is not None
            else DEFAULT_SAMPLING_SEED
        )
    return cls(
        temperature=sp.temperature,
        top_k=sp.top_k,
        top_p=sp.top_p,
        min_p=sp.min_p,
        seed=seed,
    )


# sampling.py 片段 2：sample_tokens 入口与过滤链（后续从略）
```

```
def sample_tokens(
    last_logits: mx.array,
    params: list[MlxSamplingParams],
    positions: list[int],
    key: mx.array,
    scaled: mx.array | None = None,
) -> mx.array:
    """为每一行选一个 token，全程 mx 算子，结果留在懒图内。"""
    batch_size, vocab_size = last_logits.shape
    logits32 = last_logits.astype(mx.float32)
    if scaled is None:
        scaled = scale_by_temperature(logits32, params) # logits / temperature

    # 只有需要 top-k / top-p / min-p 过滤的行才走排序链；仅调温度
    # 的行直接对 scaled logits 做 Gumbel-max —— softmax 的 logsumexp
    # 是每行常数，argmax 对其不变，省掉两个全词表 pass。
    filtering = any(
        not p.is_greedy and (p.top_k < vocab_size or p.top_p < 1.0 or p.min_p > 0.0)
        for p in params
    )
    # candidates 为 None 表示对整个词表做 Gumbel-max；否则是 [B, K]
    # 候选数组，argmax 结果再索引回真实 token id。
    candidates: mx.array | None = None
    if filtering:
        probs = mx.softmax(scaled, axis=-1)
        width = _candidate_width(params, vocab_size)
        sorted_idx = mx.argsort(-probs, axis=-1)[:, :width]
        p_sort = mx.take_along_axis(probs, sorted_idx, axis=-1)
        ranks = mx.arange(width, dtype=mx.int32)[None, :]
        # SamplingParams 把 top_k=-1 归一化为 TOP_K_ALL、把
        # temperature < eps 改写为 (temperature=1, top_k=1)，所以
        # min(top_k, width) 恒 >= 1：rank 0 总是存活，
        # log(weights) 不会全为 -inf。
        top_ks = mx.array([min(p.top_k, width) for p in params], dtype=mx.int32)[:, None]
        top_ps = mx.array([p.top_p for p in params], dtype=mx.float32)[:, None]
        min_ps = mx.array([p.min_p for p in params], dtype=mx.float32)[:, None]
        cum = mx.cumsum(p_sort, axis=-1)
        # 后续：构造 rank / 核采样 / min_p 掩码；当所有行的 top_k 都
        # 小于 MAX_BOUNDED_TOP_K=1024 时在 [B, K] 上继续（候选外权重
        # 为 0，log 为 -inf，全词表 argmax 不可能选中），否则 scatter
        # 回全词表空间；seeded 行的 Gumbel 噪声来自
        # murmur hash(seed, position, token_id)，与 CUDA kernel 同公式。
```

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

新增 WindowedAttentionKVCache（窗口有界 SWA KV 存储）并优化 make_attention_mask 的窗口无法绑定短路，是 Part 1 的核心实现。

```
# attention_kv_cache.py：WindowedAttentionKVCache 的原地追加核心
class WindowedAttentionKVCache:
```

```
    """滑动窗口层 KV 缓冲：保留尾随 window 个 token 加在途 chunk。"""

    __slots__ = ("keys", "values", "offset", "window", "_local")

    def _append(self, keys: mx.array, values: mx.array) -> tuple[int, int]:
        """原地追加一个 chunk，返回它服务的 (start, end) 区间。

        从 update_and_fetch 中拆出来，是为了让 decode 路径跳过构建
        两个返回切片 —— 它的唯一调用方 MLXAttentionWrapper._batched_decode
        只用 get_kv()。
        """
        S = keys.shape[2]
        kept = min(self._local, self.window)
        # capacity 取 window + max(S, window)：窗口本身加上正在写入的 chunk
        capacity = self.window + max(S, self.window)
        held = self.keys.shape[2] if self.keys is not None else 0
        if self._local + S > held or held > capacity:
            # 把尾窗口压缩进一个大小刚好的缓冲区：首次写入时分配，
            # 缓冲区写满时丢弃历史（每个 decode token 摊还 O(1)），
            # 过大的 prefill chunk 过去之后再缩回 2 * window。
            B, n_kv_heads, _, head_dim = keys.shape
            new_k = mx.zeros((B, n_kv_heads, capacity, head_dim), dtype=keys.dtype)
            new_v = mx.zeros((B, n_kv_heads, capacity, head_dim), dtype=keys.dtype)
            if kept:
                src = slice(self._local - kept, self._local)
                new_k[:, :, :kept, :] = self.keys[:, :, src, :]
                new_v[:, :, :kept, :] = self.values[:, :, src, :]
            self.keys, self.values, self._local = new_k, new_v, kept
        start, end = self._local - kept, self._local + S
        self.keys[:, :, self._local : end, :] = keys
        self.values[:, :, self._local : end, :] = values
        self._local = end
        self.offset += S # offset 保持绝对坐标，RoPE 位置与调度簿记不变
        return start, end
```

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

MlxModelCacheLayout 完成 full/SWA 层分区与稠密 full_kv_pool_index，是共享池拆分的结构基础，也是后续 window-aware SWA 池的接缝。

```
# layout.py：MlxModelCacheLayout 的层分区初始化
def __post_init__(self) -> None:
    # 按滑动窗口把注意力层切成两类：full 层进共享池，SWA 层只保留
    # 每请求的窗口化 KV。full_kv_pool_index 是相对共享池缓冲区的
    # 稠密索引 —— SWA 层间插时它与 attention_pool_index 不再一致，
    # 融合 AOT RoPE + 池 scatter 内核必须用前者寻址，否则会写错缓冲区。
    full_indices = tuple(
        idx
        for idx in self.attention_layer_indices
        if self.layer_window_sizes.get(idx) is None
```

```
    )
    swa_indices = tuple(
        idx
        for idx in self.attention_layer_indices
        if self.layer_window_sizes.get(idx) is not None
    )
    object.__setattr__(self, "full_attention_layer_indices", full_indices)
    object.__setattr__(self, "swa_attention_layer_indices", swa_indices)
    object.__setattr__(
        self,
        "full_kv_pool_index_by_layer",
        {layer_idx: pool_idx for pool_idx, layer_idx in enumerate(full_indices)},
    )
```

评论区精华

本 PR 无正式 review 评论，核心讨论沉淀在 PR body 的实测论证与 issue 评论中：

> alexnails（#30050 issue 评论）："metal-profiler is known CI failure. merging" 唯一的两条 issue 评论是 /tag-and-rerun-ci 与这条合并确认，CI 失败被判定为已知的 metal-profiler 问题，不阻塞合并。

PR body 中两处高价值设计论证：

> 关于采样方案选型（针对 #25804 的 CPU bridge）："That design cannot reach the default MLX scheduler path: enable_overlap_mlx is true by default on MLX, the overlap loop takes its branch before event_loop_normal..." 且实测 CPU bridge 真正致命的是 "the CPU-side sort, 60-150x the cost of the sync it accompanies"——152k 宽的排序属于 GPU，而非 sync 本身。
>
> 关于 SWA 前缀命中策略："Recomputing only a trailing band is not exact: each recomputed position needs its own window of exact hidden states, and that dependency chains backwards through every layer"——实测 trailing-band 重建使 full-attention K 偏差达 2.1、贪心输出发散，因此全量重算是正确性要求而非性能妥协。

- metal-profiler CI 失败处理 (other)：确认为已知 CI 基础设施失败，不阻塞合并；无进一步代码变更。
- 采样方案选型：图内 mx 算子 vs CPU torch Sampler 桥接（#25804） (design)：采用图内 Gumbel-max 方案（纯 mx 算子、与 forward 同懒图），保留 #25804 的 CLI flag 名 --mlx-enable-sampling。
- SWA 前缀命中策略：全量重算 vs trailing-band 重建 (design)：采用全量重算，接受 " 命中无 prefill 加速 "（hit/cold 1.00-1.01）；预留 window-aware SWA 共享池接缝。

风险与影响

- 风险：主要风险集中在四类：

1. 正确性风险（已用测试锁定）：WindowedAttentionKVCache 的 mask 宽度必须恒等于返回 key 长度，test_windowed_kv_cache.py 用三层级（缓存数组、容器 forward、

batched decode）与全历史 trailing slice 逐字节对齐锁定；但 make_mask 在全上下文不可服务时抛 RuntimeError，任何漏走窗口路径的调用都会在运行期暴露。

2. 性能风险：采样在高并发（batch >= 16）下开销显著——实测 batch 64 时 +51-65%（bounded top-K 只回收约一半）；SWA 前缀命中无 prefill 加速（相对 full-attention gather 2.5x-23x 慢）；olmo3 在 4096 token 以下窗口存储比连续存储差 2 倍（window + max(chunk, window) 预分配）。
3. 功能缺口：penalties（repetition/frequency/presence）在 Metal 上无法应用（Triton 不可用），仅一次性告警；logprob_start_len 不支持并显式拒绝；确定性采样是 MLX 局部的（float32 噪声 + MLX 排序），跨后端同 seed 可能选不同 token。
4. 平台与 CI 风险：test_swa_radix_pool.py、test_windowed_kv_cache.py 以 _HAS_MLX 作为 skip 条件，CI 覆盖依赖 mlx 环境；scheduler.py、logprob_result_processor.py、xgrammar_backend.py 等公共文件的改动会影响所有后端，虽然方向是放宽而非收紧。 - 影响：影响范围明确集中在 MLX 后端（Apple Silicon）：

- 用户侧：gpt-oss 类模型在 MLX 上从 " 只能贪心 + 全长 KV" 升级为 " 窗口 KV（decode 稳态省 47%-49%）+ 完整随机采样 + logprobs + grammar 支持 "；同时获得 mps-only 修复（is_mps() and use_mlx() 收紧 carve-out，非 MLX macOS 路径恢复快速失败）。
- 系统侧：MLX 后端 KV 路径与采样路径完成架构级重构，MlxModelCacheLayout 的 full/SWA 分区与 full_kv_pool_index 成为后续 window-aware SWA 共享池的接缝；chain_safe 调度机制是通用概念，可复用于其他后端。
- 团队侧：29 个 commit 中大量独立可读的性能优化（skip logit head、drop softmax+log、bounded top-K、padding 单次构建）为后续 MLX 开发提供基准模式；公共平台修复（pin_memory 感知、cpu 分支）惠及所有无 CUDA 主机。
- 风险标记：SWA 前缀命中无加速，采样高并发下开销显著，惩罚项不支持，新后端核心路径，确定性采样跨后端不一致，olmo3 短序列 2x 退化

关联脉络

- PR #30050 [MLX] Support gpt-oss: sliding-window attention, attention sinks, sm_scale: 本 PR 的直接构建基础：30050 使 gpt-oss 数值正确（读时加窗），本 PR 将窗口落到存储层并补齐采样；head 分支也继承自 30050。
- PR #30091 [MLX] Windowed per-request KV cache for sliding-window layers: 被本 PR 取代：窗口化 KV 缓存方案并入本分支并扩展到 radix/pool 路径，作者声明 close。
- PR #30156 [MLX] Window-bounded SWA KV storage follow-up: 同样被本 PR 取代（supersedes），其窗口化存储与池拆分方案合并演进。
- PR #25804 [MLX] CPU-bridge sampling: 采样方案被本 PR 取代：body 详细论证 CPU bridge 无法与默认 overlap 调度共存，且 CPU-side sort 成本过高；CLI flag 名沿用。
- PR #33423 CUDA sampling: clamp u=1 in gumbel noise: PR body 说明 _gumbel_noise 的 u=1 clamp 与 #33423 在 CUDA 路径修复的 +inf 确定性风险同源，两者独立落地并达成一致。