

PR #55624 完整报告

vllm-project/vllm

[V1][Metrics] Support Sliding Window Attention (SWA) and hybrid layers in MFU/MBU estimation

合并时间: 2026-09-15 00:04

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/55624>

执行摘要

- 一句话: 为 MFU/MBU 引入滑窗与混合层记账, 修正 SWA 模型指标高估
- 推荐动作: 值得精读, 尤其是做性能看板、容量规划或 v1 metrics 维护的同学。重点看三处:
 - : (1) SlidingWindowAttentionParser.parse() 的三支层类型判定及其启发式边界 (Gemma 2 硬编码、layer_types 缺失时的默认「全 SWA」); (2) AttentionMetrics 中「全局层数先拆分、再各自除以 pp_size」的不变量设计, 以及它顺带取消的向下取整对稠密模型数值口径的影响 (建议确认浮点层数在 dict[str, int] 契约下是否安全); (3) ExecutionContext 同时保留聚合标量与逐请求明细的双轨设计——快路径零回归、窗口路径才回放明细, 是低风险扩展指标能力的好范式。建议后续补一个与真实 profiler 计数对拍的 e2e 校验, 并确认是否为 MLA + SWA 混合模型补齐同样的记账。

功能与动机

issue #38170 报告在 8xH20 上跑 Step-3.5-Flash 时 `--enable-mfu-metrics` 给出平均 MFU 55.81%, 而 MiniMax-M2.5 只有 32.61%, 差距异常大; issue 作者定位到两点: MFU 统计「没有考虑 Sliding Window Attention (SWA)」和「没有考虑 Multi-Token Prediction (MTP)」, 并判断 Step-3.5-Flash 的 MFU 高于实际值。PR body 进一步量化: SWA 层中 token 最多只看 `min(context_len, sliding_window)` 个 token, 32k/128k 上下文配 4096 窗口时注意力核心 FLOPs 被高估 8-30 倍, decode 阶段 KV-cache 读流量同样被高估, 直接扭曲 MFU/MBU; 同时该 PR 也顺带解决 `perf.py` 中「TODO: discern cases where we have mixture of different attention layer types such as SWA, MLA, etc.」这条历史遗留。

实现拆解

1. 变更入口: 解析链扩展。AttentionMetrics.get_parser() 返回的 ParserChain 在 BaseAttentionConfigParser() 与 AttentionQuantizationConfigParser() 之间插入新的 SlidingWindowAttentionParser(), 顺序为 AttentionDetectionParser → BaseConfigParser → BaseAttentionConfigParser → SlidingWindowAttentionParser → AttentionQuantizationConfigParser。AttentionMetrics 的 pydantic 模型同时删掉原 TODO 注释, 新增 sliding_window: int | None = Field(None) 与 num_swa_layers: int = Field(0, ge=0) 两个字段, component_type() 仍返回 "attn", 注册表键不受影响。
2. 上下文层: ExecutionContext 增加逐请求明细与窗口截断。dataclass 新增 requests: list[tuple[int, int, bool]] = field(default_factory=list), add() 在累加原有聚合量的同时追加 (num_tokens, context_len, is_prefill); 新增

total_token_context_product(sliding_window=None)、prefill_context_len_for_window()、decode_context_len_for_window() 三个方法。当 sliding_window is None 时直接返回原有聚合标量（快路径、零回归），仅在传入窗口时回放逐请求明细并做 min(context_len, sliding_window) 截断，因此非 SWA 模型既不变行为也不付额外遍历成本。

3. 配置解析：SlidingWindowAttentionParser.parse() 三支判定层类型。窗口大小先取 model_config.get_sliding_window()，为空再回退 vllm_config.cache_config.sliding_window，且 model_config.disable_sliding_window 为真时直接按无窗口处理；层数拆分分三档——显式 hf_text_config.layer_types 列表时统计名字含 "sliding" 的层；model_type == "gemma2" 时按交替结构取 (num_layers + 1) // 2 (26 层 → 13 层)；其余有窗口的模型 (Mistral、Qwen2.5) 视为全部层 SWA。该判定结果写回 args.sliding_window / args.num_swa_layers，经 model_validate 进入 AttentionMetrics。
4. 指标计算：按层比例拆分 AttentionMetrics。get_num_flops_breakdown() 与 get_read_bytes_breakdown() 里把原来的 TC = ctx.total_token_context_product() 拆成 TC_full = ctx.total_token_context_product()（不截断）与 SWA 层使用的窗口截断值；层数从全局拆分为 L_full = L - num_swa_layers 与 L_swa = num_swa_layers 后，per-GPU 时各自除以 pp_size（而不是先把 L //= pp_size 再拆分），保证 L_full + L_swa == L。qkv_proj、out_proj 以及新生成 token 的 KV 写入仍按全部 L 层计费，只有 attn_qk、attn_av 与 attn_input 读流量按两种 context 规模加权混合，既保住投影项不被误减，也让核心项随窗口收敛。
5. 测试与配套。tests/v1/metrics/test_perf_metrics.py 为 MockModelConfig / create_mock_vllm_config() 增加 disable_sliding_window 参数，新增 9 个用例覆盖：无 SWA 默认、Mistral 全层 SWA、显式 layer_types、Gemma 2 交替 (26→13)、disable_sliding_window 关闭、FLOPs 窗口钳制 (8192 vs 1024 恰为 8 倍)、decode KV 读流量钳制、50%/50% 混合层解析式校验、以及 TP/PP 下的缩放。perf.py 内多个 parser 与抽象方法顺带补齐了 docstring (component_type、get_num_flops_breakdown、ParserChain.parse 等)，属于同批次的文档化清理。

关键文件：

- vllm/v1/metrics/perf.py（模块 性能指标；类别 source；类型 core-logic；符号 ExecutionContext.add, ExecutionContext.total_token_context_product, ExecutionContext.prefill_context_len_for_window, ExecutionContext.decode_context_len_for_window）：唯一源码文件，承载 SWA/ 混合层记账的全部核心逻辑：ExecutionContext 窗口截断、SlidingWindowAttentionParser 层类型判定、AttentionMetrics 的 FLOPs/ 读流量比例拆分，并解决了文件内关于混合注意力类型的 TODO。
- tests/v1/metrics/test_perf_metrics.py（模块 指标测试；类别 test；类型 test-coverage；符号 MockModelConfig.init, create_mock_vllm_config, test_sliding_window_parser_no_swa, test_sliding_window_parser_uniform_swa）：新增 9 个 SWA 相关单测并为 mock 配置加上 disable_sliding_window，覆盖层类型判定三支、窗口钳制比例 (8 倍) 与 TP/PP 缩放；审阅提出的 6 层 / 1 SWA / pp_size=2 回归用例也落在这里，是本 PR 语义正确性的主要护栏。

关键符号: ExecutionContext.add, ExecutionContext.total_token_context_product, ExecutionContext.prefill_context_len_for_window, ExecutionContext.decode_context_len_for_window, SlidingWindowAttentionParser.parse, AttentionMetrics.get_parser, AttentionMetrics.get_num_flops_breakdown, AttentionMetrics.get_read_bytes_breakdown, AttentionMetrics._layer_counts, create_mock_vllm_config

关键源码片段

vllm/v1/metrics/perf.py

唯一源码文件，承载 SWA/ 混合层记账的全部核心逻辑: ExecutionContext 窗口截断、SlidingWindowAttentionParser 层类型判定、AttentionMetrics 的 FLOPs/ 读流量比例拆分，并解决了文件内关于混合注意力类型的 TODO。

ExecutionContext 的窗口截断能力（新增逐请求明细 + 三个窗口方法，未传窗口时走原有聚合标量快路径）：

```
@dataclass
class ExecutionContext:
    """一次批调度的统计上下文，分别聚合 prefill / decode 两个阶段。"""

    # ... 既有聚合字段（prefill/decode 的 token 数、context 数和乘积）保持不变 ...

    # 新增：逐请求明细 (num_tokens, context_len, is_prefill),
    # 只有需要按滑动窗口截断时才会回放这批记录
    requests: list[tuple[int, int, bool]] = field(default_factory=list)

    def add(self, num_tokens: int, context_len: int, is_prefill: bool) -> None:
        """把一个请求的统计加入批上下文。"""
        self.requests.append((num_tokens, context_len, is_prefill)) # 明细与聚合同步累加
        if is_prefill:
            self.num_prefill_requests += 1
            self.prefill_num_tokens += num_tokens
            self.prefill_context_len += context_len
            self.prefill_token_context_product += num_tokens * context_len
        else:
            self.num_decode_requests += 1
            self.decode_num_tokens += num_tokens
            self.decode_context_len += context_len
            self.decode_token_context_product += num_tokens * context_len

    def total_token_context_product(self, sliding_window: int | None = None) -> int:
        """所有请求 num_tokens * context_len 之和；给窗口时对 context_len 截断。"""
        if sliding_window is None:
            # 快路径：直接用累计标量，等价于改动前实现，非 SWA 模型零回归
            return (
                self.prefill_token_context_product + self.decode_token_context_product
            )
```

```

# 慢路径：逐请求把 context_len 钳到 sliding_window，再乘 num_tokens
return sum(
    num_tokens * min(context_len, sliding_window)
    for num_tokens, context_len, _ in self.requests
)

def prefill_context_len_for_window(self, sliding_window: int | None = None) -> int:
    """prefill 请求的 context_len 之和，给窗口时按窗口截断。"""
    if sliding_window is None:
        return self.prefill_context_len
    return sum(
        min(context_len, sliding_window)
        for _, context_len, is_prefill in self.requests
        if is_prefill
    )

def decode_context_len_for_window(self, sliding_window: int | None = None) -> int:
    """decode 请求的 context_len 之和，给窗口时按窗口截断（用于 KV 读流量）。"""
    if sliding_window is None:
        return self.decode_context_len
    return sum(
        min(context_len, sliding_window)
        for _, context_len, is_prefill in self.requests
        if not is_prefill
    )

```

tests/v1/metrics/test_perf_metrics.py

新增 9 个 SWA 相关单测并为 mock 配置加上 `disable_sliding_window`，覆盖层类型判定三分支、窗口钳制比例（8 倍）与 TP/PP 缩放；审阅提出的 6 层 / 1 SWA / `pp_size=2` 回归用例也落在这里，是本 PR 语义正确性的主要护栏。

代表性用例：窗口生效时核心 attention FLOPs 被钳住而投影项保持不变，以及 mock 配置新增的 `disable_sliding_window` 开关：

```

class MockModelConfig:
    def __init__(self, hf_config, dtype, disable_sliding_window=False):
        """新增开关，用于构造 --disable-sliding-window 场景。"""
        ...
        self.disable_sliding_window = disable_sliding_window

def test_attention_metrics_sliding_window_flops_bounds():
    """context 超过窗口时，核心 attention FLOPs 被钳到窗口，投影项不受影响。"""
    sw = 1024
    dense_config = MistralConfig(
        hidden_size=2048, num_attention_heads=16, num_key_value_heads=16,
        num_hidden_layers=8, sliding_window=None,
    )
    swa_config = MistralConfig(

```

```

hidden_size=2048, num_attention_heads=16, num_key_value_heads=16,
num_hidden_layers=8, sliding_window=sw,
)
dense_metrics = AttentionMetrics.from_vllm_config(
    create_mock_vllm_config(dense_config)
)
swa_metrics = AttentionMetrics.from_vllm_config(create_mock_vllm_config(swa_config))

# 1) context 未超过窗口时，两者数值必须完全一致（零回归）
short_ctx = ExecutionContext.from_single_request(
    num_tokens=1, context_len=512, is_prefill=False
)
assert swa_metrics.get_num_flops(short_ctx) == dense_metrics.get_num_flops(short_ctx)

# 2) context 8192 > 窗口 1024: 核心项恰好缩小 8 倍，投影项不变
long_ctx = ExecutionContext.from_single_request(
    num_tokens=1, context_len=8192, is_prefill=False
)
swa_flops = swa_metrics.get_num_flops_breakdown(long_ctx)
dense_flops = dense_metrics.get_num_flops_breakdown(long_ctx)
assert swa_flops["qkv_proj"] == dense_flops["qkv_proj"]
assert swa_flops["out_proj"] == dense_flops["out_proj"]
assert swa_flops["attn_qk"] * 8 == dense_flops["attn_qk"]
assert swa_flops["attn_av"] * 8 == dense_flops["attn_av"]

# 另有一组解析器用例校验层类型判定：
# - Gemma 2: 26 层中 13 层走滑窗（偶数层），见 test_sliding_window_parser_gemma2
# - 显式 layer_types 混合：6 层中 3 层滑窗，见 test_sliding_window_parser_explicit_layer_types
# - disable_sliding_window=True: sliding_window 归 None、num_swa_layers 归 0

```

评论区精华

核心交锋集中在流水并行下的层数拆分。markmc 借助 Codex 给出 P1：

"For a 6-layer model with one SWA layer and `pp_size=2`, `L` becomes 3 and `round(3 * 1 / 6)` becomes 0. The estimate therefore treats all three layers as full attention, misreporting both attention FLOPs and `attn_input` reads."

即先用 `L //= pp_size` 再按比例拆分会把少数派 SWA 层舍成 0。作者 thillai-c 复现后采纳第一种方案（从全局层数拆分再各自除以 `pp_size`），并解释为何不做 PP-rank 感知：

"these metrics are constructed by `from_vllm_config()` from config alone, with no pipeline rank available at that point. So the per-GPU number here represents the average pipeline stage rather than a specific stage."

修复后 6 层 / 1 SWA / PP=2 得到 2.5 个全注意力层与 0.5 个 SWA 层。CodeRabbit 另提出两条：per-GPU 拆分需保持 `L_full + L_swa == L` 不变量，以及补一个 hybrid + `pp_size>1` 的回归用例——作者在 commit [95e08e5](#) 一并处理，并更新了原有编码旧 8/7 拆分的测试。此外

markmc 提醒按 contributing 指南标注 AI 协助，作者随后在 PR body 补上说明并承诺后续 commit 使用 **Assisted-by: trailer**; markmc 最终 APPROVED。

- 流水并行下混合层模型的 SWA 层被舍入成 0 (P1) (correctness): 作者复现确认后采纳「从全局层数拆分、再各自除以 pp_size」的方案 (commit 95e08e5)，保证 $L_{full} + L_{swa} == L$ ，该形状下得到 2.5 个全注意力层与 0.5 个 SWA 层；未采用 PP-rank 感知方案，理由是 from_vllm_config() 只有配置、拿不到 pipeline rank，现值为流水 stage 平均值。
- per-GPU 拆分需保持 $L_{full} + L_{swa} == L$ 不变量 (correctness): 最终实现改为从全局层数拆分再各自除以 pp_size (作者说明这也替换了原先 $L // pp_size$ 的向下取整)，并新增 6 层 / 1 SWA / pp_size=2 的回归测试，同时更新了原先把旧 8/7 拆分写死的 hybrid PP 用例。
- 补充 hybrid 层 + pipeline_parallel_size > 1 的测试覆盖 (testing): 作者按 P1 修复补上了 6 层 / 1 SWA / pp_size=2 的回归测试，并同步更新原有 hybrid PP 断言，覆盖该组合。
- AI 辅助贡献的署名与透明度 (documentation): 作者在 PR body 补充说明 (Claude Code 协助、逐行复核并自行运行测试)，并承诺后续 commit 使用 Assisted-by: trailer; 由于修复提交已在历史中，未重写其 commit message。该实践随后被维护者接受并 APPROVED。

风险与影响

- 风险:
 1. 近似边界未与真实 kernel 计数对齐。SWA 层用 $\text{num_tokens} * \min(\text{context_len}, \text{sliding_window})$ 记账，忽略 prefill 三角边界 (首个 token 只能看自己) 与窗口按 block 对齐的差异，是上界近似；对长上下文相对误差小，但没有任何与 profiler/nsys 实际计数对拍的 e2e 校验，单元测试只验证公式自治。
 2. PP 口径变化可能波及非 SWA 模型。修复把「先 $L // pp_size$ 再拆分」改成「全局层数各自除以 pp_size」 (作者明确说明这同时替换了原先的向下取整)，于是 per-GPU 层数可能变成小数 (如 2.5 / 0.5)，对层数不能被 pp_size 整除的稠密模型也会改变数值口径；而 get_num_flops_breakdown() 的返回注解仍是 dict[str, int]，浮点值是否能被下游 Prometheus 指标路径安全消费需要复核。
 3. 层类型判定含硬编码启发式。model_type == "gemma2" 分支写死交替模式，其他同为交替结构但 layer_types 缺失的模型会落到「全部层都是 SWA」分支，反而系统性低估注意力开销；"sliding" in lt.lower() 也依赖 HF 侧字符串命名约定，未来重命名会静默失效。
 4. MLA 路径未同步。SWA 字段与拆分只加在 AttentionMetrics，MLAAttentionMetrics 保持原状，若混合配置下 MLA 与滑窗并存，相关模型指标仍会失真；MTP 记账在 issue #38170 中也已声明单独跟踪。
 5. ExecutionContext 逐请求列表的开销。requests 会在每个 batch 的每次 add() 中追加元组，在 --enable-mfu-metrics 打开且高频 step 下增加少量分配与内存；量级远小于模型计算，但属于常驻结构变化。- 影响：对用户：开启 --enable-mfu-metrics 的部署中，Mistral、Qwen2.5、Gemma 2、Step-3.5-Flash 等 SWA/ 混合层模型的 MFU、MBU 会从明显偏离的高位回落到接近真实水平，长上下文任务尤其明显；由于数字本身变小，跨版本的历史看板与容量规划基线不可直接对比，需要重新标定。对系统：改动只

落在 vllm/v1/metrics/perf.py 的指标估算路径，不触碰调度、KV cache 管理与 attention kernel，推理正确性与性能不受影响；但解析异常会以 InvalidComponent 形式让 AttentionMetrics 整块跳过，属于既有的容错语义。对团队：为后续 MTP 记账、MLA 滑窗记账以及「PP-rank 精确语义」预留了扩展点（当前 per-GPU 值代表流水 stage 平均值而非某一 rank 的真实构成），同时该 PR 也确立了在 vLLM 中显式标注 AI 协助贡献的实践样例。- 风险标记：PP 口径变化，层类型判定启发式，近似上界未对拍，MLA 路径未覆盖，指标基线需重标

关联脉络

- PR #56791 [Bugfix] Trim stale consequence claims from unannotated-eagle warning: 同样围绕 v1 中「混合注意力层类型」（滑窗层与全注意力层并存的模型）的配置解读与诊断信息做小步修正，且同样以源码 + 针对性单测（tests/v1/core/test_kv_cache_utils.py）的方式落地，与本 PR 对 num_swa_layers/ 层类型判定的处理属同一主题脉络。
- PR #50894 [Bugfix] Scale KV page size for hidden states extraction with TP: 同属 v1 引擎中「并行维度（TP/PP）下的规格与口径换算」类修复：该 PR 处理 TP 下 KV page size 放大以避免启动断言崩溃，本 PR 处理 pp_size 下层数拆分导致 SWA 层被舍入；两者都以显式计算替代隐含假设并补单元测试。
- PR #56016 [CPU][Profiler] Group torch profiler tables by input shape when record_shapes is on: 同一时期对性能可观测性数据口径的修正：该 PR 让 profiler 表格按输入形状分组以提升统计可读性，本 PR 修正 MFU/MBU 的解析式统计口径，二者都服务于「让性能数字可信」这一方向。