PR #34819 完整报告

sgl-project/sglang

config: the post-publish consumers of the supplied-instance surface read the bags

合并时间：2026-08-15 15:39

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34819

执行摘要

- 一句话：配置读取从 server_args 迁移到 bags，覆盖 17 个消费点
- 推荐动作：值得精读。它是 sglang 配置系统迁移（supplied-instance → bags）的样板 PR：转换边界划分清晰（四批 + 保留清单），每批都有明确理由；对"何时读 bag、何时读 record"给出了可复用的判别标准（post-publish 纯消费读 bag；build-from-record 契约与 publish 前路径保留）。特别值得关注 ModelRunner publish 上移的时点论证和 dspark worker 的自相矛盾读取案例。建议结合 #34267（暴露面钉测）和 #34269（bag 契约测试）一起读，理解整个迁移栈的闭环。

功能与动机

PR body 明确指出这是 step-12 配置迁移的一部分：supplied-instance surface 上的读取应改为从已发布的 bags 读取，否则 ServerArgs 一旦持有用户原始输入，读到已解析字段的调用方会看到 raw 值而非 effective 值。PR 提到一个关键反例：dspark worker 在实例读取 server_args.page_size 下方三十行却读 get_exec().graph.cuda_graph_config.decode.bs，同一文件对同一值来源自相矛盾。另有一个 Codex 捕获的真实崩溃：ModelRunner 独立构造（python -m sglang.benchmark.one_batch、手动 runner 测试）时 constructor 自身的 publish位于本次读取之下，触发ValueError:configgnamespace'schedule'notpublished。

实现拆解

实现按四批转换推进，每批对应 disposition 表的一行：

1. 七个 speculative worker 构造器的 page_size 读取：dspark_worker_v2.py、frozen_kv_mtp_worker_v2.py、dflash_worker_v2.py、multi_layer_eagle_worker_v2.py、ngram_worker.py、standalone_worker_v2.py、eagle_worker_v2.py 中 self.page_size = server_args.page_size 改为 get_schedule().page_size。这些 worker 都在 publish 之后运行，且只是拷贝一个进程级值，因此是纯消费转换。
2. post-publish 的 chunked_prefill_size 消费者：expert_distribution.py（EPLB recorder 缓冲大小）、compile_utils.py 的 update_deep_gemm_config（deep-gemm 编译 warmup 的五处读取）、kv_cache_builder.py 的 build_kv_cache（effective size）、ngram_embedding_manager.py 的 NgramEmbeddingManager.from_model（assert）全部改为 get_schedule().chunked_prefill_size；deep-gemm warmup 顺带把五次重复读取收敛为每个分支绑定一次局部变量（review nit 要求）。

3. graph/limit 消费者：dspark_worker_v2.py 的 DsparkVerifyEpilogue(max_bs=...) 改读 get_exec().graph.cuda_graph_config.decode.bs，dspark_planner.py 的 SPS table bound（max_running_requests）经 build_sps_cost_table 的调用方测试改为通过 bag override 提供，lora_manager.py 的 init_lora_cuda_graph_moe_buffers 改读 get_exec().graph...。
4. 三个进程级构造器的 page_size：model_runner.py、scheduler.py、decode_kvcache_offload_manager.py；ModelRunner.__init__ 中 set_global_server_args_for_scheduler(server_args) 从构造函数尾部上移到 self.page_size = get_schedule().page_size 之前（仅 target worker，is_draft_worker 分支不变），保证独立构造也能先发布再读取；offload manager 顺带删除了对 server_args 的整对象驻留。

测试配套：test/registered/spec/dspark/test_dspark_sps_table.py 的 _build_sps_cost_table_for 不再注入 SimpleNamespace 伪记录，而是通过 get_context().override_server_args(...) 将 speculative_dspark_sps_table_path 和 max_running_requests 发布到 bags，再取 get_server_args() 传给 build_sps_cost_table，并在 addCleanup 中恢复 override，确保 override 不泄漏到其他测试。

刻意保留不转换的：契约是"从交给你的 record 构建 X"的工厂（create_kt_config_from_server_args、DllmConfig.from_server_args、CanaryLaunchCapacities.from_args、build_compilation_config）、resolution pipeline 以 resolved_view 调用的 helper（utils/common 的 topk/page 谓词）、由调用方传入配置的 allocation_sizing、per-instance 边界（CudaVmmFeatureTransport、DP controller、encode-server 家族）以及历史上先于 publish 运行的 initialize_moe_config。

关键文件：

- python/sglang/srt/model_executor/model_runner.py（模块 模型执行；类别 source；类型 data-contract；符号 ModelRunner.init, set_global_server_args_for_scheduler）：核心构造器：page_size 读取从 server_args 改为 get_schedule()，且 set_global_server_args_for_scheduler 从构造尾部上移到首次 bag 读取之前，修复独立构造（benchmark/one_batch、手动 runner 测试）时的 not-published 崩溃；publish 时点调整是全 PR 最敏感的控制流变更。
- python/sglang/srt/layers/deep_gemm_wrapper/compile_utils.py（模块 编译预热；类别 source；类型 dependency-wiring；符号 update_deep_gemm_config）：deep-gemm 编译 warmup 的五处 chunked_prefill_size 读取全部从 server_args 切换到 get_schedule()，并按 review nit 收敛为每个分支一次绑定，避免重复读 bag。
- test/registered/spec/dspark/test_dspark_sps_table.py（模块 测试；类别 test；类型 test-coverage；符号 _build_sps_cost_table_for）：唯一的测试配套改动：_build_sps_cost_table_for 从注入 SimpleNamespace 伪记录改为通过 get_context().override_server_args(...) 发布到 bags，并在 addCleanup 恢复，验证了本 PR 的消费方转换在测试中的正确接线方式。
- python/sglang/srt/speculative/dspark_worker_v2.py（模块 推测解码；类别 source；类型 dependency-wiring；符号 DSparkWorkerV2.init）：PR body 亲自点名的"自相矛盾"案例：page_size 改读 bag 的同时，三十行之下的 DsparkVerifyEpilogue(max_bs=...) 也从

server_args.cuda_graph_config 改为 get_exec().graph.cuda_graph_config，统一了同一文件内的值来源。

- python/sglang/srt/speculative/frozen_kv_mtp_worker_v2.py（模块 推测解码；类别 source；类型 dependency-wiring；符号 FrozenKVMTPDraftWorker.init, FrozenKVMTPWorkerV2.init）：FrozenKVMTPDraftWorker 与 FrozenKVMTPWorkerV2 两个构造器的 page_size 都从 server_args 切到 get_schedule()，是 speculative worker 批量转换的代表性文件。
- python/sglang/srt/model_executor/model_runner_components/ngram_embedding_manager.py（模块 嵌入管理；类别 source；类型 data-contract；符号 NgramEmbeddingManager.from_model）：ngram embedding 的 assert 从 server_args.chunked_prefill_size 改为 get_schedule().chunked_prefill_size，属于 post-publish chunked_prefill_size 消费批次。
- python/sglang/srt/disaggregation/decode_kvcache_offload_manager.py（模块 KV 卸载；类别 source；类型 dependency-wiring；符号 DecodeKVCacheOffloadManager.init）：decode 侧 KV offload manager 的 page_size 改读 bag，并顺带删除对 server_args 整对象驻留，减少 supplied record 的引用面。
- python/sglang/srt/lora/lora_manager.py（模块 LoRA；类别 source；类型 dependency-wiring；符号 LoRAManager.init_lora_cuda_graph_moe_buffers）：LoRA 管理器的 cuda-graph MoE buffers 的 max_bs 改从 get_exec().graph.cuda_graph_config.decode.max_bs 读取，属于 graph/limit 消费批次。
- python/sglang/srt/mem_cache/kv_cache_builder.py（模块 KV 缓存；类别 source；类型 dependency-wiring；符号 build_kv_cache）：KV cache 构建器的 effective chunked_prefill_size 改读 get_schedule()，是 post-publish 消费批次的一部分。
- python/sglang/srt/speculative/dflash_worker_v2.py（模块 推测解码；类别 source；类型 dependency-wiring；符号 DFlashWorkerV2.init）：DFlash worker 构造器的 page_size 改读 get_schedule()，是七个 speculative worker 批量转换之一。

关键符号：ModelRunner.init, update_deep_gemm_config, NgramEmbeddingManager.from_model, DSparkWorkerV2.init, FrozenKVMTPDraftWorker.init, FrozenKVMTPWorkerV2.init, DFlashWorkerV2.init, build_kv_cache, init_lora_cuda_graph_moe_buffers, DecodeKVCacheOffloadManager.init, _build_sps_cost_table_for

关键源码片段

python/sglang/srt/model_executor/model_runner.py

核心构造器：page_size 读取从 server_args 改为 get_schedule()，且 set_global_server_args_for_scheduler 从构造尾部上移到首次 bag 读取之前，修复独立构造（benchmark/one_batch、手动 runner 测试）时的 not-published 崩溃；publish 时点调整是全 PR 最敏感的控制流变更。

```
# python/sglang/srt/model_executor/model_runner.py
# __init__ 开头部分的关键时序调整：
# 旧代码在构造函数尾部才调用 set_global_server_args_for_scheduler，
```

```
    # 而 page_size 读取发生在其之前；当 ModelRunner 被独立构造时
    # （python -m sglang.benchmark.one_batch、手动 runner 测试）
    # 没有更早的 publish，会触发 ValueError: config namespace 'schedule' not published。
    # 新代码把发布动作上移，确保任何 bag 读取前都已发布。

    def __init__(
        self,
        model_config: ModelConfig,
        mem_fraction_static: float,
        gpu_id: int,
        ps: ParallelState,
        nccl_port: int,
        server_args: ServerArgs,
        is_draft_worker: bool = False,
        req_to_token_pool: Optional[ReqToTokenPool] = None,
        token_to_kv_pool_allocator: Optional[BaseTokenToKVPoolAllocator] = None,
        memory_pool_config: Optional[MemoryPoolConfig] = None,
        draft_model_idx: Optional[int] = None,
        draft_attention_backend: Optional[str] = None,
    ):
        # ... 前置字段赋值省略 ...

        # 提前设置 scheduler 进程的全局 server_args（仅 target worker），
        # 放在构造器最早期的 bag 读取之前：独立构造没有更早的 publish 来源。
        if not is_draft_worker:
            set_global_server_args_for_scheduler(server_args)

        # ... 中间初始化省略 ...

        # page_size 从已发布的 schedule bag 读取，而不是从传入的 supplied record 读取；
        # 这样 post-publish 的 override 能像影响其他消费者一样影响本构造器。
        self.page_size = get_schedule().page_size
```

python/sglang/srt/layers/deep_gemm_wrapper/compile_utils.py

deep-gemm 编译 warmup 的五处 chunked_prefill_size 读取全部从 server_args 切换到 get_schedule()，并按 review nit 收敛为每个分支一次绑定，避免重复读 bag。

```
    # python/sglang/srt/layers/deep_gemm_wrapper/compile_utils.py
    # update_deep_gemm_config 的 warmup 分支：
    # 原先五处读取都从 server_args（supplied record）取 chunked_prefill_size，
    # 现改为从已发布的 schedule bag 读取，并在每个分支开头绑定一次局部变量，
    # 避免对同一 bag 的重复读取（review nit）。

    def update_deep_gemm_config(gpu_id: int, server_args: ServerArgs):
        global _BUILTIN_M_LIST
        global _DO_COMPILE_ALL
        global _IS_FIRST_RANK_ON_NODE

        _BUILTIN_M_LIST = []
```

```
    if _FAST_WARMUP:
        # 快速预热：覆盖 decode 常用小 batch，再按步长采样覆盖大 batch。
        _BUILTIN_M_LIST += list(range(1, 1025))

        next_m, sample_step = 1024, 2
        # 每个分支只读一次 bag，后续计算复用局部变量。
        chunked_prefill_size = get_schedule().chunked_prefill_size
        max_prefill_bs = (
            min(chunked_prefill_size, 32 * 1024)
            if chunked_prefill_size >= 1
            else 16 * 1024
        )
        while next_m < max_prefill_bs:
            _BUILTIN_M_LIST += list(range(next_m, 2 * next_m, sample_step))
            next_m = next_m * 2
            sample_step = sample_step * 2
        _BUILTIN_M_LIST.append(max_prefill_bs)
        _BUILTIN_M_LIST = sorted(list(set(_BUILTIN_M_LIST)))
    else:
        # 非快速预热：按 m_max 覆盖全部 M；chunked_prefill_size 决定上限。
        m_max = 1024 * 16
        chunked_prefill_size = get_schedule().chunked_prefill_size
        if chunked_prefill_size < 1:
            m_max = 1024 * 64
        elif chunked_prefill_size > 8192:
            m_max = chunked_prefill_size * 2
        m_max = min(1024 * 128, m_max)
        _BUILTIN_M_LIST += list(range(1, m_max + 1))

    _IS_FIRST_RANK_ON_NODE = server_args.base_gpu_id == gpu_id
    _DO_COMPILE_ALL = _IS_FIRST_RANK_ON_NODE
```

test/registered/spec/dspark/test_dspark_sps_table.py

唯一的测试配套改动：_build_sps_cost_table_for 从注入 SimpleNamespace 伪记录改为通过 get_context().override_server_args(...) 发布到 bags，并在 addCleanup 恢复，验证了本 PR 的消费方转换在测试中的正确接线方式。

```
# test/registered/spec/dspark/test_dspark_sps_table.py
# 被测函数 build_sps_cost_table 现在从已发布的 bags 读取 max_running_requests，
# 因此不能再像以前那样注入 SimpleNamespace 伪记录；
# 测试改为通过 override 机制发布真实配置，并在用例结束后恢复，
# 避免 override 泄漏到同进程的其他测试。

def _build_sps_cost_table_for(testcase, *, sps_table_path):
    from sglang.srt.runtime_context import get_context, get_server_args
    from sglang.srt.speculative.dspark_components.dspark_planner import (
        build_sps_cost_table,
    )
```

```
    # 表格上界读取的是已发布的 bag 值，所以测试发布它；
    # table 路径仍由 build_sps_cost_table 从传入的 record 读取，保持不变。
    override = get_context().override_server_args(
        speculative_dspark_sps_table_path=sps_table_path,
        max_running_requests=4,
    )
    override.install()
    testcase.addCleanup(override.restore)
    return build_sps_cost_table(
        server_args=get_server_args(), verify_num_draft_tokens=5
    )
```

评论区精华

本 PR 无 GitHub review 评论（review_comments_count=0），讨论主要沉淀在 PR body 与被替换的 #34268 旧线程中。可从 body 提炼的要点包括：

- dspark worker 自相矛盾的读取：文件内三十行之下已读 *get_exec*().graph.cuda_graph_config.decode.bs，上方却读 server_args.page_size，是同一值来源不一致的最清晰例证。
- review nit：deep-gemm warmup 原本五次重读 bag，现改为每个分支绑定一次局部变量。
- Codex 捕获的独立构造崩溃：ModelRunner 独立构造（benchmark/one_batch、手动 runner 测试）时 constructor publish 位于读取之下，触发 ValueError: config namespace 'schedule' not published，本 PR 将 publish 上移。
- 刻意保留清单：PR body 明确列出保留不转换的工厂 /helper 及其理由（契约是 build from record、运行在 publish 之前等）。
  - dspark worker 内同一值来源自相矛盾 (design): 两类读取统一改为从 bags 读取：page_size 走 get_schedule()，graph 配置走 *get_exec*()。
  - deep-gemm warmup 重复读取 bag (style): 每个分支绑定一次局部变量，后续计算复用。
  - ModelRunner 独立构造时 not-published 崩溃 (correctness): set_global_server_args_for_scheduler 上移到构造器早期、首次 bag 读取之前；scheduler 路径不受影响。
  - 刻意保留不转换的工厂与 helper (design): 这些读取按契约保留在 record 上，不纳入本批转换。
  - 测试改用 override 发布而非 SimpleNamespace 伪记录 (testing): 采用真实发布 + 恢复的接线方式，override 不泄漏出测试。

风险与影响

- 风险：主要风险点：
  - publish 时点依赖：ModelRunner 中 set_global_server_args_for_scheduler 上移到构造器早期，若未来有人在该调用之前新增 bag 读取（如 enable_hisparse 等仍在下方），会再次触发 not-published 崩溃。当前 page_size 读取已在 publish 之后，但 ModelRunner 中其余 server_args.* 读取（如 elastic_ep_backend、enable_hisparse）

未一并转换，存在部分读取走 record、部分走 bag 的混合态。

- 覆盖不彻底：census 共 314 对，本 PR 只转换 17 对，剩余 297 对靠 #34267 的 ratchet 钉住；任何漏网读取在 ServerArgs 持有 raw 输入后可能静默读到未解析值。
- 测试泄漏风险：test_dspark_sps_table.py 改用 override.install() 发布全局状态，虽用 addCleanup(override.restore) 保证恢复，但若 override 机制本身有异常路径，可能污染同进程其他测试。
- bag 未发布即读取：若存在跳过 scheduler publish 的启动路径（除已知的 standalone ModelRunner 外），get_schedule() 会抛错；PR 已修复已知路径，但未全面枚举所有启动入口。

- 影响：影响范围：
  - 对用户：无用户可见行为变化，纯内部配置读取路径重构；唯一功能语义变化是 post-publish override 现在能影响此前从 record 读取的 17 个消费点，这正是预期目标。
  - 对系统：涉及 speculative decoding（EAGLE、DFlash、DSpark、Frozen-KV MTP、ngram、standalone）、MoE、LoRA、KV cache 构建、EPLB 调度、deep-gemm 编译 warmup、disaggregation offload 等多个子系统，覆盖面广但每个改动都是单一字段的来源替换，回归面可控。
  - 对团队：为 step-12 配置迁移扫清了生产侧障碍，使后续 #34267 的暴露面钉测能以最终形态落位；同时确立了“进程级值读 bags、record 契约读 record”的处置规范，后续新增代码有章可循。
  - 风险标记：配置读取路径重构，publish 时点敏感，混合读取态残留，测试依赖全局 override

关联脉络

- PR #34267 config: pin the supplied-instance surface that a raw record would change: 同一迁移栈的下游成员：本 PR 的生产转换 + 剩余 297 对的钉测。PR body 明确说明本 PR 先于 #34267 合并，钉测列表在 #34819 落地后以最终形态落位。
- PR #34269 config: state the bag contract as what resolution produced, and the skill rule that goes with it: 同一 step-12 迁移栈的文档与契约测试成员，为 bag 读取语义提供契约基线。
- PR #34913 [CI] Move the static ratchets back to CPU unit tests: 将静态 ratchet 检查移回 CPU 单元测试，与本 PR 及 #34267 的钉测机制配套，形成 CI 闭环。