PR #34376 完整报告

sgl-project/sglang

[Fix] Make the linear-attn kernel choice per-runner, and pin draft/target loader-hook parity

合并时间：2026-08-15 15:36

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34376

执行摘要

- 一句话：linear-attn 内核选择改为 per-runner 隔离，修复 draft/target 配置污染
- 推荐动作：值得精读。重点看三个设计决策：per-runner stamp 替代进程级表；fail-closed（去掉静默 triton 回退）的守卫方式；draft/target hook parity 不变量测试如何用『存在性一致』而非『实现一致』来表达『必须回答 loader 的问题』。此外 Codex 的 P1 泄漏机制讨论很有价值，它解释了为什么把自动默认值写进进程级配置叶子是泄漏源——建议结合 test_the_per_runner_default_stays_out_of_the_process_config 一起阅读。

功能与动机

PR body 将本 PR 定位为 #33312 的延续：#33312 修复的是一种『per-runner 决策中两个参与者答案不一致』的问题——DSV4 DSpark draft entry 类没有暴露其 target 家族携带的 shared-experts-fusion gate，loader 为 target 和 draft 装了不同决策，draft 权重按错误布局装载，accept length 从 5.60 掉到 2.05。本 PR 修复同类问题：linear-attn 内核后端是进程级共享的，attn_backend_wrapper 每 runner 重建一次模块级字典，导致 draft 无法持有与 target 不同的选择，且 config-override 路径不写回 ServerArgs 记录，第二次重建会把第一次的选择静默覆盖为基础后端。此外旧的 test_fusion_gate_coverage.py 只能抓住『已消费决策』的类，抓不住『应当消费却没有消费』的类，需要新的不变量测试。

实现拆解

1. 解析入口替换：linear/utils.py 删除进程级 _BACKENDS 字典、initialize_linear_attn_config()、_get_backend() 与三个模块级 getter，新增 frozen LinearAttnBackends（msgspec.Struct）与纯函数 resolve_linear_attn_backends()。新函数从 get_exec().mamba 读取已发布的配置叶子，prefill 优先级为「显式 flag > 调用方传入的 auto-default > 基础 backend」，verify 未设置时沿用 decode 的 flashinfer 分支或回退 triton，语义与旧路径等价。
2. 打 stamp 的位置：attention_registry.py 的 attn_backend_wrapper 在构造 linear-attn 后端对象之前执行 runner.linear_attn_backends = resolve_linear_attn_backends(prefill_default=...)；同时删除原先通过 get_context().override("gdn_backend.sm100_flashinfer_default", ...) 把自动默认值写进进程级配置的路径——这正是泄漏机制：第二个 runner 会把 target 的自动默认值误读成操作员显式 flag。
3. 消费者改造：GDN（gdn_backend.py）、KDA（kda_backend.py）、Ascend GDN（ascend_gdn_backend.py）三个后端构造函数的读取源从模块级 getter 改为

model_runner.linear_attn_backends 的 decode / prefill / verify 字段。后端构造发生在 stamp 之后，因此未打 stamp 的构造路径会因属性缺失直接抛 AttributeError，fail-closed，不再存在『工作但错误』的静默 triton 回退。

4. 测试与夹具配套：test_linear_attn_config.py 重写为 TestLinearAttnBackends，钉住优先级、双 runner 各自持有不同选择、unstamped runner 在真实 GDNAttnBackend 构造时抛错、以及自动默认值不进入进程配置；新增 test_draft_entry_hook_parity.py 对注册表中每个名字带 NextN / MTP / DSpark / DFlash / Standalone 后缀的 draft entry class 断言 shared_experts_fusion_disable_reason 存在性与 target 一致；三个 attention fixture（gdn_attention.py、kda_attention.py、lightning_attention.py）改为模拟 attn_backend_wrapper 先打 stamp 再构造后端。另清理了 utils.py 中不再使用的 logging 导入与 logger 绑定。
5. 影响面：单 runner 启动行为不变；目标场景（GDN target + 非 GDN draft 共存于同一进程）从『draft 权重按错误布局装载』修复为各自持有独立选择。

关键文件：

- python/sglang/srt/layers/attention/linear/utils.py（模块 注意力配置；类别 source；类型 core-logic；符号 LinearAttnBackends, resolve_linear_attn_backends, initialize_linear_attn_config, _get_backend）：核心文件：进程级 _BACKENDS 表替换为 per-runner LinearAttnBackends stamp，新增 resolve_linear_attn_backends()，是本次修复的语义载体；同时清理了静默 triton 回退与死代码。
- python/sglang/srt/layers/attention/attention_registry.py（模块 注意力注册；类别 source；类型 dependency-wiring）：stamp 的落点：attn_backend_wrapper 在构建后端前设置 runner.linear_attn_backends，并删除把自动默认值写入进程级配置的 get_context().override(...) 路径，这是修复泄漏机制的关键。
- test/registered/unit/models/test_draft_entry_hook_parity.py（模块 钩子对等；类别 test；类型 test-coverage；符号 _target_of, TestDraftEntryHookParity, test_a_draft_resolves_a_gate_exactly_when_its_target_does）：新增不变量测试：对注册表中每个名字带 NextN / MTP / DSpark / DFlash / Standalone 后缀的 draft entry class 断言 shared_experts_fusion_disable_reason 存在性与 target 一致，能拦住 #33312 那类『draft 未暴露 target 携带的门控』缺陷。
- test/registered/unit/layers/attention/test_linear_attn_config.py（模块 配置单测；类别 test；类型 test-coverage；符号 TestLinearAttnBackends, _Runner, _publish, test_an_unstamped_runner_raises_rather_than_guessing）：重写后的单测：钉住 flag > 默认值 > base 的优先级、双 runner 各持不同选择、unstamped runner 在真实 GDNAttnBackend 构造时抛 AttributeError、以及自动默认值不进入进程配置。
- python/sglang/srt/layers/attention/linear/gdn_backend.py（模块 GDN 后端；类别 source；类型 core-logic；符号 GDNAttnBackend.init）：GDN 后端构造函数从模块级 getter 改读 model_runner.linear_attn_backends，是三个消费者之一。
- python/sglang/srt/layers/attention/linear/kda_backend.py（模块 KDA 后端；类别 source；类型 core-logic；符号 KDAAttnBackend.init）：KDA 后端构造函数同步改读 stamp，且 EAGLE tree-verify 限制依赖 verify backend 判定，属于敏感消费点。

- python/sglang/srt/hardware_backend/npu/attention/ascend_gdn_backend.py（模块 NPU 后端；类别 source；类型 dependency-wiring；符号 AscendGDNAttnBackend.init）：NPU 上 Ascend GDN 后端的同步改造，保证 NPU 平台与 CUDA 平台行为一致。
- python/sglang/test/kits/attention_unittest/attention_methods/gdn_attention.py（模块 测试夹具；类别 test；类型 test-coverage；符号 build_gdn_attention_fixture）：测试夹具模拟 attn_backend_wrapper 先打 stamp 再构造后端，防止 fixture 因缺 stamp 而失败。
- python/sglang/test/kits/attention_unittest/attention_methods/kda_attention.py（模块 测试夹具；类别 test；类型 test-coverage；符号 build_kda_attention_fixture）：KDA 测试夹具同步适配 stamp 路径。
- python/sglang/test/kits/attention_unittest/attention_methods/lightning_attention.py（模块 测试夹具；类别 test；类型 test-coverage；符号 build_lightning_attention_fixture）：Lightning attention 测试夹具同步适配 stamp 路径。

关键符号：resolve_linear_attn_backends, LinearAttnBackends, GDNAttnBackend.init, KDAAttnBackend.init, AscendGDNAttnBackend.init, attn_backend_wrapper, _target_of, TestDraftEntryHookParity.test_a_draft_resolves_a_gate_exactly_when_its_target_does

关键源码片段

python/sglang/srt/layers/attention/linear/utils.py

核心文件：进程级 _BACKENDS 表替换为 per-runner LinearAttnBackends stamp，新增 resolve_linear_attn_backends()，是本次修复的语义载体；同时清理了静默 triton 回退与死代码。

```
# linear/utils.py —— 每个 runner 的 linear-attn 内核选择
#
# 旧实现用进程级 `_BACKENDS` 字典，每次 runner 构建都重建一次；
# 第二次重建会通过 config-override 路径把第一次的选择覆盖掉
#  （override 不写回 ServerArgs 记录，导致后续 runner 解析回基础 backend）。
# 新实现把选择封装成 per-runner 的不可变 stamp，由 wrapper 在构建后端前打好。

class LinearAttnBackends(msgspec.Struct, frozen=True):
    """单个 runner 的 linear-attn 内核选择，按阶段分开存放。

    target 与 draft 共存在同一进程，可以持有不同选择：只有模型是 GDN 的
    runner 才会得到 SM100 FlashInfer prefill 默认值，显式 flag 也只作用于
    启动时传入它的那个 runner。
    """

    decode: LinearAttnKernelBackend
    prefill: LinearAttnKernelBackend
    verify: LinearAttnKernelBackend


def resolve_linear_attn_backends(
    prefill_default: Optional[str] = None,
) -> LinearAttnBackends:
```

```
    """从已发布的配置叶子解析出本 runner 的内核选择。

    `prefill_default` 是调用方自己的自动默认值（SM100 GDN 场景）；
    显式配置的 `--linear-attn-prefill-backend` 优先级更高。
    解析完成后返回 frozen 容器，只打日志，不写入任何进程级配置。
    """
    mamba = get_exec().mamba # 已发布的配置叶子，只含启动时显式请求的值
    base = mamba.linear_attn_backend
    decode = LinearAttnKernelBackend(mamba.linear_attn_decode_backend or base)
    prefill = LinearAttnKernelBackend(
        mamba.linear_attn_prefill_backend or prefill_default or base
    )

    # verify 未设置时沿用历史行为：decode 是 flashinfer 则用其
    # recurrent kernel，否则回退 triton。
    verify = mamba.linear_attn_verify_backend
    if verify is None:
        verify = decode.value if decode.is_flashinfer() else "triton"

    backends = LinearAttnBackends(
        decode=decode, prefill=prefill, verify=LinearAttnKernelBackend(verify)
    )
    rank0_log(
        f"Linear attention kernel backend: decode={backends.decode.value}, "
        f"prefill={backends.prefill.value}, verify={backends.verify.value}"
    )
    return backends
```

python/sglang/srt/layers/attention/attention_registry.py

stamp 的落点：attn_backend_wrapper 在构建后端前设置 runner.linear_attn_backends，并删除把自动默认值写入进程级配置的 get_context().override(...) 路径，这是修复泄漏机制的关键。

```
# attention_registry.py —— attn_backend_wrapper 中的 stamp 逻辑
# 在构建任何 linear-attn 后端对象之前，先把本 runner 的内核选择打上去；
# 后端构造函数读取的就是这个 stamp。删除旧的
# `get_context().override("gdn_backend.sm100_flashinfer_default", ...)`
# 记录路径，避免 target 的自动默认值泄漏成进程级“显式 flag”。

check_environments()
prefill_default = None
# 只有模型为 GDN 且非 NPU 的 runner 才带 SM100 FlashInfer 自动默认值
if hybrid_gdn_config(runner.model_config) is not None and not is_npu():
    prefill_default = flashinfer_gdn_prefill_default(runner)

# 先 stamp，再构造后端；未走此路径的后端会因属性缺失立即抛出
# AttributeError（fail-closed），而不是静默回退到“工作但错误”的内核。
runner.linear_attn_backends = resolve_linear_attn_backends(
    prefill_default=prefill_default
```

```
)

hybrid_backend_cls = HybridLinearAttnBackend
if hybrid_gdn_config(runner.model_config) is not None:
    # 后续分支用 runner.prefill_attention_backend_str /
    # decode_attention_backend_str 校验 Blackwell / NPU 上允许的后端组合
    ...
```

test/registered/unit/models/test_draft_entry_hook_parity.py

新增不变量测试：对注册表中每个名字带 NextN / MTP / DSpark / DFlash / Standalone 后缀的 draft entry class 断言 shared_experts_fusion_disable_reason 存在性与 target 一致，能拦住 #33312 那类『draft 未暴露 target 携带的门控』缺陷。

```
# test_draft_entry_hook_parity.py —— draft 与 target 的 loader-hook 存在性对等
# 背景：loader 会向 entry class 询问 shared-experts-fusion 决策
# （install_shared_experts_fusion_decision）；若 draft 的 entry class 没有
# 暴露 target 家族携带的自动关闭条件，draft 就会解析出与 target 不同的
# 决策，权重按错误的布局装载。DSV4 DSpark 曾因此缺陷引入回归
# （accept length 从 5.60 掉到 2.05，由 PR #33312 修复）。

# draft 后缀：`...NextN` / `...MTP` / `...DSpark` / `...DFlash` / `...Standalone`
# 对应的模型通常是 target 的某个阶段镜像；`...Eagle` 是独立 checkpoint，不在此列。
DRAFT_SUFFIX = re.compile(r"(NextN|MTP|DSpark|DFlash|Standalone)$")

# 刻意不检查 weight-name maps：draft checkpoint 有自己的名字，天然不同。
PARITY_HOOKS = ("shared_experts_fusion_disable_reason",)


def _target_of(arch: str, archs: dict):
    """返回 draft entry class 所服务的 target 架构名，若它按名字指向 target。"""
    match = DRAFT_SUFFIX.search(arch)
    if not match:
        return None
    base = arch[: match.start()]
    for candidate in (base, f"{base}ForCausalLM"):
        if candidate in archs and candidate != arch:
            return candidate
    return None


class TestDraftEntryHookParity(CustomTestCase):
    def test_a_draft_resolves_a_gate_exactly_when_its_target_does(self):
        # 遍历注册表，对每一对 draft/target 断言 hook 存在性一致；
        # 只比较"有没有"，不要求实现相同（Qwen3.5 MTP 带参数委托是合法的）。
        checked, offenders = 0, []
        for arch, cls in archs.items():
            target = _target_of(arch, archs)
            if target is None:
                continue
```

```
            for hook in PARITY_HOOKS:
                checked += 1
                if hasattr(cls, hook) == hasattr(archs[target], hook):
                    continue
                offenders.append(f"{arch} vs {target}: {hook} 存在性不一致")
        self.assertGreater(checked, 10, "draft/target 配对未找到任何条目")
        self.assertEqual([], offenders, "draft 与 target 必须对 loader hook 给出相同答案")
```

评论区精华

核心讨论围绕 4 个问题展开，全部解决：

- *Codex* 的 P1：自动默认值泄漏。*Codex* bot 指出旧代码通过 *get_context().override(...)* 把 SM100 GDN 的 *linear_attn_prefill_backend="flashinfer"* 写进进程级配置，后续非 GDN draft 解析时会读到该叶子，即使 *prefill_default=None* 也会被错打上 FlashInfer stamp，per-runner 隔离并未真正生效。作者承认并修复：删除该 *override* 路径，让配置叶子只承载启动时显式请求的值，自动默认值只存在于 runner 自己的 stamp 中，并用测试双向钉住隔离。
- *msgspec* 约定。评审建议 LinearAttnBackends 使用 *msgspec.Struct*（frozen）而非 *dataclasses.dataclass*，符合项目对新容器的规则。作者已转换，并指出 frozen 同时强化了『stamp 一次后不可变更』的意图。
- 测试同义反复。原 *test_an_unstamped_runner_raises* 只断言空类缺属性，任何空类都成立；作者改为构造最小 runner double 并调用真实 GDNAttnBackend(runner)，断言 AttributeError 信息包含 linear_attn_backends，让失败确实来自生产代码的 guard。
- 死代码清理。移除 *_get_backend* 后遗留的 *import logging* 与 *logger* 绑定被删除。
- 自动默认值泄漏进进程级配置导致 draft 仍读到 FlashInfer (correctness): 作者确认这是泄漏机制并修复：删除 *get_context().override(...)* 记录路径，让配置叶子只承载启动时显式请求的值，自动默认值只存在于 runner 自己的 stamp 中；*test_linear_attn_config.py* 新增双向钉住该隔离。
- 新容器必须用 *msgspec.Struct* 而非 *dataclass* (design): 作者已转换，并指出 frozen 也能强化『stamp 一次后不可变更』的意图。
- unstamped runner 测试是空类上的同义反复 (testing): 作者改为构造最小 runner double（CPU 设备 + conv pool），调用真实 GDNAttnBackend(runner) 并断言 AttributeError 信息包含 linear_attn_backends，让失败确实来自生产代码的 guard。
- 删除回退后遗留死代码 logging/logger (style): 作者已删除，并说明该条此前位于 review body 的『outside the diff』区，现已纳入检查通道。

风险与影响

- 风险：
  1. 动态 stamp 属性：runner.linear_attn_backends 是动态添加的属性，未在 ModelRunner 类定义中显式声明。任何绕过 attn_backend_wrapper 的后端构造路径（如未来的新 fixture 或硬件后端）都会立即抛 AttributeError；这是设计意图（fail-closed），但对扩展者是行为变更。

2. 配置语义变化：get_context().override(...) 记录路径被删除后，resolved_server_args_dict() 不再包含自动默认的 linear_attn_prefill_backend 字段，若有其他模块依赖该字段观察 SM100 默认值会读到 None。测试已钉住『自动默认值不进入进程配置』，但对外部观测者属于语义变化。
3. verify 后端等价性：verify 默认逻辑（decode.value 为 flashinfer 时跟随 decode，否则 triton）与旧实现等价，但 KDA 的 EAGLE tree-verify 限制（topk > 1 时抛 ValueError）依赖 verify backend 判定，任何解析偏差都会在 KDAAttnBackend.__init__ 暴露。
4. CI 状态：PR 页记录的 Base / Extra 两次 CI 运行均为失败状态，作者声明 CPU 单元测试无新增失败，但合并前仍需确认失败原因与本 PR 无关。 - 影响：对用户与系统：修复 GDN target + 非 GDN draft 组合（如 DSV4 DSpark 与 GDN target 同进程）下 draft 内核选择被静默覆盖的问题，避免权重布局错误导致接受率从 5.60 掉到 2.05 这类退化；单 runner KDA / GDN / NPU Ascend 用户行为不变。对配置语义：进程级配置叶子只承载启动时显式请求的值，自动默认值改由 per-runner stamp 承载，与 full-attn 后端的 prefill_attention_backend_str / decode_attention_backend_str 模式对齐。对团队：确立了 per-runner stamp 的配置承载方式，以及『draft 与 target 必须对 loader hook 给出相同答案』的不变量测试，后者能让 #33312 一类缺陷在未来提交中被自动拦截。 - 风险标记：进程级配置覆盖路径已删除，per-runner stamp 为动态属性未在类中声明，涉及 GDN/KDA/Ascend 多后端协同，CI 状态显示失败需确认，配置语义变化影响外部观测者

关联脉络

- PR #33312 （前置修复，PR body 引用，上下文未提供标题）：PR body 将本 PR 定位为 #33312 修复的同一形状问题的延续：DSV4 DSpark draft 未暴露目标家族的 shared-experts-fusion gate，导致 accept length 从 5.60 掉到 2.05。本 PR 的 test_draft_entry_hook_parity.py 正是为钉住该类缺陷而新增。
- PR #34264 [config] decisions keyed on the attention backend read the configured pair: 同属注意力后端决策读取语义修正：该 PR 让基于注意力后端的决策读取 prefill/decode 配置对，本 PR 让 linear-attn 内核选择变成 per-runner stamp，两者共同完善注意力后端配置读取模型。
- PR #34263 [config] the last runner-side instance reads read the bags: 本 PR 的 resolve_linear_attn_backends 从 get_exec().mamba 读取配置叶子，属于 config bag 迁移系列；该系列 PR 一直在把 server_args 直读迁移到已发布配置。
- PR #34265 [config] a named entry point for the resolution pipeline, and the last dynamic config read: config bag 系列的先导 PR，命名了配置解析管线入口；本 PR 的解析函数与之一脉相承，且同样涉及 get_context().override 路径的清理。
- PR #32593 [Kernel] Enable Helion backend for Kimi Delta-Attention: 同一 kda_backend.py 演进线上：该 PR 为 KDA 引入 Helion 后端并扩展了 server_args 配置，本 PR 则把这些配置的读取方式从进程级表改为 per-runner stamp。