PR #36725 完整报告

sgl-project/sglang

config: every handler declares its cuda-graph decisions

合并时间：2026-08-28 03:55

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36725

执行摘要

- 一句话：cuda-graph 决策改由各 handler 显式声明
- 推荐动作：值得精读，尤其是 with_phase 的设计（不可变更新原语）和 TestDeclaredValuesAreNotEditedLater 的 guard 测试写法（包装全部 stash 写入通道 + 多形态矩阵）。由于该 PR 无独立 review 讨论，建议与系列后续 PR #36620-#36622 一起 review，以理解声明模型的完整演进。阅读时重点关注 _handle_gpu_memory_settings 的 deepcopy 例外和 with_phase 的共享引用警告。

功能与动机

PR body 明确指出问题所在：_parse_cuda_graph_config 声明了一个 CudaGraphConfig，但 19 个后续 handler 通过 resolving_view 拿到该对象并原地编辑（58 处赋值）。“错误在于 provenance：overrides log 把决策记在 _parse_cuda_graph_config 头上，而 validate_declarations 永远看不到后来的修改。”换言之，值是对的，但归属错了，导致配置来源不可追踪、声明校验失效。这属于配置解析基础设施的健壮性修正。

实现拆解

1. 引入不可变更新原语 with_phase：在 python/sglang/srt/model_executor/cuda_graph_config.py 新增 with_phase(config, phase, **changes)，用 dataclasses.replace 重建两个 PhaseConfig 并构造新的 CudaGraphConfig，确保结果与输入不共享任何 PhaseConfig。这是所有后续声明改写的基础，避免一个 handler 的修改污染另一个 handler 的声明。
2. 将 18 个 handler 的 36 处原地赋值改为声明：在 python/sglang/srt/server_args.py 各 _handle_* 方法中，把 cfg.cuda_graph_config.decode.backend = ... 这类写入改写为 self._declare("_handler名", cuda_graph_config=with_phase(...))。典型如 _handle_model_capability_adjustments 中 HRM-Text、EmbeddingGemma 和 prefill sizing 分支，每个决策以 handler 自身名义进入 stash，修复来源归属。
3. _handle_gpu_memory_settings 特殊处理：该 handler 承担 58 个赋值中的 22 个，且彼此依赖并读取自身早期赋值，因此保留对 cfg.cuda_graph_config 的 deepcopy，在全部计算结束后统一声明一次。
4. 加固 guard 测试：在 test/registered/unit/server_args/test_resolution_declarations.py 新增 TestDeclaredValuesAreNotEditedLater，包装 declare_resolution、declare_late_resolution、declare_direct_writes、run_post_process_pass 四个 stash 写入通道，在每条记录落地时深拷贝，结束时对比；覆盖 8 种启动形态（plain、cuda-graph

旋钮、chunked prefill、显式 JSON、disaggregation、deterministic、EAGLE、DPattention）。

5. 更新既有测试与计数核查：test/registered/unit/server_args/test_server_args.py 的 test_decode_graph_stays_enabled_in_both_comm_modes 改用 resolution_result 读取声明后的配置；用 AST pass 重新统计赋值面（19 方法 58 赋值），修正窄 grep 漏项。

关键文件：

- python/sglang/srt/server_args.py（模块 参数解析；类别 source；类型 core-logic；符号 _handle_model_capability_adjustments, _handle_gpu_memory_settings, _parse_cuda_graph_config）：主变更文件，19 个 handler 中大部分 cuda-graph 原地赋值改为 self._declare + with_phase 声明，修复 provenance 归属并影响所有启动流程。
- test/registered/unit/server_args/test_resolution_declarations.py（模块 声明守卫；类别 test；类型 test-coverage；符号 TestDeclaredValuesAreNotEditedLater, _resolve_recording_each_entry, test_no_entry_changes_after_it_is_recorded）：新增 guard 测试 TestDeclaredValuesAreNotEditedLater，包装四个 stash 写入通道并在每条记录落地时深拷贝，捕获“声明后被修改”的漂移；覆盖 8 种启动形态，是验证本次重构正确性的核心测试。
- python/sglang/srt/model_executor/cuda_graph_config.py（模块 图配置；类别 source；类型 data-contract；符号 with_phase）：新增 with_phase 不可变更新函数，是本次重构的核心原语，决定了所有 handler 如何声明新的 cuda_graph_config。
- python/sglang/srt/hardware_backend/npu/utils.py（模块 NPU 后端；类别 source；类型 dependency-wiring；符号 set_default_server_args）：NPU 后端默认参数设置从局部别名解码改为 with_phase 声明，是外部 handler 迁移的代表，涉及 Ascend 多型号的 max_bs 决策。
- test/registered/unit/server_args/test_server_args.py（模块 参数测试；类别 test；类型 test-coverage）：既有断言从直接读属性改为读取 resolution_result，适配新的声明形态，确保测试验证的是解析后的最终值。

关键符号：with_phase, _handle_model_capability_adjustments, _handle_gpu_memory_settings, set_default_server_args, _resolve_recording_each_entry, test_no_entry_changes_after_it_is_recorded, _parse_cuda_graph_config

关键源码片段

python/sglang/srt/server_args.py

主变更文件，19 个 handler 中大部分 cuda-graph 原地赋值改为 self._declare + with_phase 声明，修复 provenance 归属并影响所有启动流程。

```
# 在 _handle_model_capability_adjustments 内，HRM-Text (prefix_lm) 检测分支。
# 变更前直接改 cfg.cuda_graph_config.decode.backend / prefill.backend，
# 变更后改为以本 handler 名义声明新的配置，确保 provenance 归属正确。
if is_hrm_text and getattr(hf_config, "prefix_lm", True):
    run_post_process_pass(self, _hrm_text_attention_force)
    self._declare(
```

```
        "_handle_model_capability_adjustments",
        chunked_prefill_size=-1,
    )
    self._declare(
        "_handle_model_capability_adjustments",
        disable_radix_cache=True,
    )
    self._declare(
        "_handle_model_capability_adjustments",
        disable_cuda_graph=True,
    )
    # cuda_graph_config 已从 legacy boolean 解析，仅翻转 boolean 不会停止
    # graph capture，因此必须直接声明 backend 为 DISABLED。
    self._declare(
        "_handle_model_capability_adjustments",
        cuda_graph_config=with_phase(
            cfg.cuda_graph_config, Phase.DECODE, backend=Backend.DISABLED
        ),
    )
    self._declare(
        "_handle_model_capability_adjustments",
        cuda_graph_config=with_phase(
            cfg.cuda_graph_config, Phase.PREFILL, backend=Backend.DISABLED
        ),
    )
```

test/registered/unit/server_args/test_resolution_declarations.py

新增 guard 测试 TestDeclaredValuesAreNotEditedLater，包装四个 stash 写入通道并在每条记录落地时深拷贝，捕获“声明后被修改”的漂移；覆盖 8 种启动形态，是验证本次重构正确性的核心测试。

```
class TestDeclaredValuesAreNotEditedLater(CustomTestCase):
    """声明记录的是值，不是句柄。

    stash 保留 handler 传入的对象；如果 handler 声明一个可变对象后又原地
    修改它，就会重写已经进入日志的条目。projection 仍然回答最终状态，
    所以其他机制不会察觉——丢失的是“哪个 handler 决定了什么”，
    validate_declarations 也看不到后续修改。
    """

    def _resolve_recording_each_entry(self, **supplied):
        """解析，并在每条 stash 条目落地的那一刻深拷贝一份。"""
        from sglang.srt.arg_groups import overrides

        recorded = []

        def watch(name):
            original = getattr(overrides, name)
```

```
        def wrapper(server_args, *args, **kwargs):
            result = original(server_args, *args, **kwargs)
            stash = getattr(server_args, "_resolved_overrides", None) or []
            while len(recorded) < len(stash):
                index = len(recorded)
                recorded.append((index, copy.deepcopy(stash[index])))
            return result

        return original, wrapper

    # 所有会追加到 stash 的通道。
    patched = {}
    for name in (
        "declare_resolution",
        "declare_late_resolution",
        "declare_direct_writes",
        "run_post_process_pass",
    ):
        original, wrapper = watch(name)
        patched[name] = original
        setattr(overrides, name, wrapper)
    try:
        path = tempfile.mkdtemp(prefix="declared_values_")
        self.addCleanup(shutil.rmtree, path, ignore_errors=True)
        with open(os.path.join(path, "config.json"), "w") as handle:
            json.dump(_MINI_CONFIG, handle)
        server_args = ServerArgs(
            model_path=path, device="cuda", random_seed=42, **supplied
        )
        server_args.resolve_once()
    finally:
        for name, original in patched.items():
            setattr(overrides, name, original)
    return server_args, recorded

def test_no_entry_changes_after_it_is_recorded(self):
    # 每个 handler family 取一种启动形态。
    for label, supplied in (
        ("plain", {}),
        ("cuda_graph_knobs", {"cuda_graph_max_bs_decode": 16}),
        ("chunked_prefill", {"chunked_prefill_size": 1024}),
        ("explicit_json", {"cuda_graph_config": {"decode": {"max_bs": 12}}}),
        ("disaggregation", {"disaggregation_mode": "prefill"}),
        ("deterministic", {"enable_deterministic_inference": True}),
        ("speculative", {"speculative_algorithm": "EAGLE"}),
        ("dp_attention", {"tp_size": 2, "dp_size": 2, "enable_dp_attention": True}),
    ):
        with self.subTest(shape=label):
            server_args, recorded = self._resolve_recording_each_entry(**supplied)
```

```
stash = server_args._resolved_overrides
self.assertGreater(len(recorded), 0, "没有记录，说明没在监视")
drifted = [...] # 对比 recorded 与最终 stash
self.assertEqual(drifted, [], "声明被后续修改")
```

python/sglang/srt/model_executor/cuda_graph_config.py

新增 with_phase 不可变更新函数，是本次重构的核心原语，决定了所有 handler 如何声明新的 cuda_graph_config。

```
def with_phase(config: "CudaGraphConfig", phase: str, **changes) -> "CudaGraphConfig":
    """返回只修改一个 phase 的 config 副本。

    Resolution 语义是"声明值"，而不是"编辑句柄"：handler 决定某个图配置时，
    交给 stash 的必须是一个新对象，而不是修改先前 handler 已声明的对象。
    因此这里重建两个 phase，保证返回的 config 与输入不共享任何 PhaseConfig。
    """
    if phase not in Phase.ALL:
        raise KeyError(phase)
    # 不是深拷贝：dataclasses.replace 复制字段引用，所以 list 型字段 bs
    # 仍然共享。调用方必须 rebind bs，绝不能原地 mutate 它。
    return CudaGraphConfig(
        **{
            # 只有目标 phase 应用 changes，另一个 phase 原样 replace。
            name: replace(getattr(config, name), **(changes if name == phase else {}))
            for name in Phase.ALL
        }
    )
```

评论区精华

PR 无外部 review 评论，以下提炼自作者 PR body 中的设计说明：

- 作者指出 provenance 问题是核心："什么错的是 provenance：overrides log 把决策记在 _parse_cuda_graph_config 头上，而 validate_declarations 永远看不到后来的修改。"
- 初始计数严重低估："那是个窄 grep，只匹配到局部别名拼写，漏掉了直接拼写……它还匹配 == 比较，直到我加了负 lookahead——把其中一个转成声明会是一个真正的 bug。"
- Guard 矩阵扩展的价值："disaggregation_mode="prefill" 和 enable_deterministic_inference=True 各到达一个前三种 shape 从未跑过的 handler，且都还在漂移。DP-attention shape 又抓到两个——通过局部别名访问 config 的 handler，第一遍转换漏掉了。"
- 对 torch.compile 的处理："object.__getattribute__ 会 graph-break。这是被测量而不是假设——新读取能追踪到 torch.compile(fullgraph=True)，并被回归测试固定。"
- 声明来源归属与 validate_declarations 盲区 (design): 引入 with_phase，让每个 handler 以自己名义声明新的 cuda_graph_config，修复来源归属。
- 初始计数低估与窄 grep 风险 (correctness): 改用 AST pass 统计，得到 19 方法 58 赋值，并加负 lookahead 排除比较。

- guard 测试矩阵扩展暴露漂移 handler (testing): 扩展到 8 种启动形态，覆盖所有 handler 家族。
- torch.compile graph-break 风险 (performance): 接受改动，以回归测试固定该行为。

风险与影响

- 风险：核心风险是启动路径变更：server_args.py 是每个服务进程必经的配置解析路径，19 个 handler 的声明方式重构可能影响特定模型 / 硬件组合（如 HRM-Text、NPU、EmbeddingGemma）的最终 cuda-graph 配置。虽然 24 种启动形态 × 478 共享字段的分辨率对比为 0 差异，但覆盖仅限 CPU 侧，未覆盖真实多进程组或多卡环境。 其次，with_phase 并非深拷贝：dataclasses.replace 复制字段引用，bs 这类 list 字段仍共享；若未来有代码原地改动 list，将引入别名 bug（代码注释已明确警告）。 此外，本 PR 无 GPU 精度测试，所有验证基于 CPU 侧 resolution dump 与 guard；且该 PR 强依赖同一系列的后续 PR（#36620-#36622），若后续回退可能破坏一致性。NPU 路径（npu/utils.py）的声明方式变化虽值相同，但缺少 NPU 实测。
- 影响：对用户：无可观察行为变化，但配置来源日志更准确，便于排查“谁改了 cuda-graph 配置”。对系统：修复 overrides 来源归属，validate_declarations 能看见后续决策；声明不可变的模式为后续平行读取重构铺路。对团队：本 PR 确立了“声明不可变、来源可追踪”的配置编写范式，是五 PR 系列的关键一步，后续 #36620-#36622 均建立在此模型上。
- 风险标记：核心配置路径变更，无 GPU 精度验证，系列前置依赖，不可变更新共享引用风险

关联脉络

- PR #36620 config: a parallel leaf with no live counterpart is read bare: 同一五 PR 系列的第三部分，本 PR 引入的声明模型为它铺路；PR body 明确要求按顺序 review。
- PR #36621 config: a parallel size has one spelling; a patched scope declares its own: 系列第四部分，基于本 PR 的声明基础设施做平行读取设计。
- PR #36622 config: the record is not an object that gets passed around: 系列第五部分，进一步消除 ServerArgs 对象传递，依赖本 PR 的声明形态。