PR #37087 完整报告

sgl-project/sglang

[Config] Round 5.2: the per-model declarations get their own modules

合并时间：2026-08-30 17:24

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37087

执行摘要

- 一句话：按模型家族拆分配置覆盖声明，守卫字段重叠
- 推荐动作：值得精读。重点关注三处设计：一是 model_override_base.py 的双视图设计——ResolvedView（构造时快照 overlay，适合 post-process pass）与 ResolvingConfig（每次读取遍历 stash，适合声明后的实时读取），以及 collect_model_override_declarations 的 last-writer-wins 语义；二是 " 用测试禁止重叠代替 pin import 顺序 " 的决策，这是对 import 副作用注册体系的正确解耦；三是 test_model_source_paths.py 的 mutation 验证方法，展示了如何证明测试真的有效而不是假设有效。后续维护者新增模型家族时应遵守 model_overrides/ 目录约定并保持单字段单一归属。

功能与动机

PR body 引用 2026-07-03 placement ruling：config-time model overrides 必须留在 arg_groups/（模型模块不在 __post_init__ 导入、前端进程不得依赖 GPU kernel 导入），但 family 增长后允许机械拆分。27 个 handler 和 992 行挤在单个 3043 行的 overrides.py 里，新增模型家族意味着要编辑所有家族共用的文件，且各家族逻辑被埋在注册表、视图和后处理 pass 之间。拆分此前被 #37086 阻塞——它给 platform facts 一个唯一地址后，family 模块才能连同 probe reads 一起移出。

实现拆解

1. 基础设施下沉到 model_override_base.py：注册表（MODEL_OVERRIDES、_MODEL_OVERRIDE_FNS、_PREDICATE_OVERRIDE_FNS）、注册装饰器（register_model_override、register_model_override_predicate、_register_for）、只读视图（ResolvedView 快照视图、ResolvingConfig 实时视图）、访问器（attention_backends_of、model_config_of、get_default_attn_backend 等）全部移入新文件。依赖方向是关键：family 模块只 import base，不反向依赖 overrides.py，因此没有循环 import。overrides.py 通过 from sglang.srt.arg_groups import model_override_base 读取注册对象并 re-export 同名符号，保持既有调用方兼容。
2. 26 个家族模块纯移动：每个模块一个 _<family>_overrides(server_args, hf_config) -> dict，用 @_register_for(...) 注册一个或多个架构，model_overrides/__init__.py 的 import 列表即注册动作。验证方式是 live registry 快照前后一致（55 个架构、61 次注册、顺序一致）以及 62-shape resolution probe 在 7 个 commit 上逐一对比 IDENTICAL，而不是逐文件人工审查。

3. 非机械决策：禁止字段重叠：isort 在拆分中自动排序 import 列表曾翻转 InternS2Mobius 的赢家声明。作者放弃固定 import 顺序，改为由 test_model_override_split.py 断言任何架构的两个注册者不得声明同一字段、常量表不与会话竞争、磁盘上的 family 模块必须全部出现在 import 列表中。字段名提取复用 chain-read ratchet 的工具函数，避免两套扫描逻辑不一致。
4. 重复规则收编：modelexpress_config 的 JSON 解析此前在 arg_groups（transfer-engine gate）和 record（_parsed_modelexpress_config 属性）各有一份，统一为 modelexpress_config_of；pre_capture_activation_reserve_mb 改写为 pre_capture_activation_reserve_mb_of（arg_groups helper + runtime_context 的 published-bag 双胞胎，tier 一致性测试改为两 tier 直接对比）；ssl_verify 并入 serving_hook 的 SSL 校验；kv_event_block_size、_support_mamba_cache_extra_buffer 等 record 成员删除并委托到 arg_groups。ServerArgs 净删约 70 行。
5. 测试与验证配套：test_model_source_paths.py 覆盖三条此前无 PR-CI 的模型路径解析轴（GGUF Hub 引用、ModelScope repo id、remote-connector URL），全部通过 mock/stub 免网络，并用三个 mutation 验证测试有效性（删 GGUF 声明失败 3 例、删 ModelScope 下载目录查找失败 1 例、删 model_weights 写入失败 1 例）；处理器测试从裸 MagicMock 改为真实 ServerArgs，避免所有属性为真值导致 worker 计数分支误判。

关键文件：

- python/sglang/srt/arg_groups/model_override_base.py（模块 覆盖注册表；类别 source；类型 data-contract；符号 register_model_override, register_model_override_predicate, _invoke_provider, ResolvedView）：PR 的核心：注册表、只读视图（ResolvedView/ResolvingConfig）与访问器全部下沉到这里，定义 family 模块的依赖方向，是拆分的地基。
- python/sglang/srt/arg_groups/overrides.py（模块 覆盖注册表；类别 source；类型 dependency-wiring；符号 collect_model_override_declarations, run_post_process_pass, pre_capture_activation_reserve_mb_of, modelexpress_config_of）：从 3043 行瘦身到 1931 行，负责 re-export 与后处理 pass/收集逻辑，是依赖接线和兼容性收口点。
- python/sglang/srt/arg_groups/model_overrides/kimi_k3.py（模块 模型配置；类别 source；类型 data-contract；符号 _require_kimi_k3_cutedsl_dcp_support, _kimi_k3_overrides, _kimi_k3_moe_runner_overrides）：26 个家族模块中最复杂的一个，集中 DCP/DSPARK/SM100 注意力与 MoE 后端默认，最能代表 per-family 声明的真实复杂度。
- python/sglang/srt/server_args.py（模块 服务参数；类别 source；类型 core-logic；符号 pre_capture_activation_reserve_mb, ssl_verify, _parsed_modelexpress_config, modelexpress_url）：Record 上 7 个派生成员被删除并委托到 arg_groups，是本次契约变更的落点，外部调用面需关注。
- test/registered/unit/test_model_override_split.py（模块 拆分守卫；类别 test；类型 test-coverage；符号 _returned_field_names, TestModelOverrideSplit, test_no_field_is_declared_by_two_family_modules,

test_the_import_list_names_every_family_module）：把 " 禁止字段重叠 " 这个非机械决策固化为测试，保证 import 顺序不再承载行为，是拆分质量的契约标尺。

- test/registered/unit/server_args/test_model_source_paths.py（模块 模型路径；类别 test；类型 test-coverage；符号 _ModelSourceCase, TestTheGgufArm, TestTheModelScopeArm, TestTheRemoteConnectorArm）：补上 GGUF/ModelScope/remote-connector 三轴 PR-CI 测试空白，并用 mutation 验证有效性，防止回归再次漏网。
- python/sglang/srt/arg_groups/model_overrides/__init__.py（模块 模型配置；类别 source；类型 entrypoint）：import 列表即注册动作；isort 曾在此翻转赢家，是守卫测试保护的入口。
- python/sglang/srt/arg_groups/model_overrides/minimax_m3.py（模块 模型配置；类别 source；类型 data-contract；符号 _minimax_m3_overrides）：per-family 模块化示例之一：HIP/SM100/SM90 的注意力后端与 MoE runner 声明，展示平台分支如何随家族迁移。
- python/sglang/srt/arg_groups/model_overrides/deepseek_v2.py（模块 模型配置；类别 source；类型 data-contract；符号 _deepseek_family_overrides）：DeepSeek/DSA 家族大模块（9 个架构），含 DSA/MLA CP 与 page_size 默认，是拆分后受益明显的文件。

关键符号：register_model_override, register_model_override_predicate, _register_for, _invoke_provider, resolving_view, resolved_view, model_config_of, collect_model_override_declarations, modelexpress_config_of, pre_capture_activation_reserve_mb_of, _kimi_k3_overrides, _kimi_k3_moe_runner_overrides, _minimax_m3_overrides, _deepseek_family_overrides

关键源码片段

python/sglang/srt/arg_groups/model_override_base.py

PR 的核心：注册表、只读视图（ResolvedView/ResolvingConfig）与访问器全部下沉到这里，定义 family 模块的依赖方向，是拆分的地基。

```
"""per-model 覆盖声明所依赖的基础设施。"""

import logging
from typing import Any, Callable, Dict, List, Optional, Tuple

logger = logging.getLogger(__name__)


# 常量级覆盖：arch -> {field: value}，在 derived provider 之前应用。
MODEL_OVERRIDES: Dict[str, Dict[str, Any]] = {
    "MistralLarge3ForCausalLM": {"dtype": "bfloat16"},
    "PixtralForConditionalGeneration": {"dtype": "bfloat16"},
}

# 派生覆盖 provider 注册表，按注册顺序保存。
_MODEL_OVERRIDE_FNS: Dict[str, List[Callable[..., dict]]] = {}
```

```
# 谓词键控 provider（旧分支按架构子串匹配），同样按注册顺序。
_PREDICATE_OVERRIDE_FNS: List[Tuple[Callable[[str], bool], Callable[..., dict]]] = []


def register_model_override(architecture: str):
    """注册一个按架构精确匹配的派生覆盖 provider。

    被装饰的可调用对象接收 (server_args, hf_config)，不得修改两者，
    返回 {field: resolved_value} 字典；什么都不适用时返回空字典。
    """

    def decorator(fn: Callable[..., dict]) -> Callable[..., dict]:
        _MODEL_OVERRIDE_FNS.setdefault(architecture, []).append(fn)
        return fn

    return decorator


class ResolvedView:
    """post-process pass 拿到的只读快照视图。

    构造时把当前 declaration stash 叠加为 overlay；pass 从自己的槽位
    读取时看到的是"已累积声明覆盖在原始字段上"的状态。写入被拒绝，
    pass 只能返回新声明。
    """

    __slots__ = ("_server_args", "_overlay")

    def __init__(self, server_args: Any, overlay: Optional[Dict[str, Any]] = None):
        object.__setattr__(self, "_server_args", server_args)
        object.__setattr__(self, "_overlay", overlay or {})

    def __getattr__(self, name: str) -> Any:
        overlay = object.__getattribute__(self, "_overlay")
        if name in overlay:
            return overlay[name]
        # overlay 没有的字段回退到 record 的原始输入
        return getattr(object.__getattribute__(self, "_server_args"), name)

    def __setattr__(self, name: str, value: Any) -> None:
        raise AttributeError("ResolvedView is read-only; passes return declarations")


class ResolvingConfig:
    """实时读取视图：每次读取都走一遍 stash，最后写入者优先。

    ResolvedView 在构造时快照 overlay，适合 post-process pass；
    而 resolver 在"声明之后"再读取时，需要的是当前最新答案。
    """
```

```
    __slots__ = ("_server_args",)

    def __init__(self, server_args: Any):
        object.__setattr__(self, "_server_args", server_args)

    def __getattr__(self, name: str) -> Any:
        server_args = object.__getattribute__(self, "_server_args")
        for _source, declared in reversed(
            getattr(server_args, "_resolved_overrides", None) or ()
        ):
            if name in declared:
                return declared[name]
        return getattr(server_args, name)

    def __setattr__(self, name: str, value: Any) -> None:
        raise AttributeError("ResolvingConfig is read-only")
```

python/sglang/srt/arg_groups/model_overrides/kimi_k3.py

26 个家族模块中最复杂的一个，集中 DCP/DSPARK/SM100 注意力与 MoE 后端默认，最能代表 per-family 声明的真实复杂度。

```
@_register_for("KimiK3ForConditionalGeneration")
def _kimi_k3_overrides(server_args: Any, hf_config: Any) -> dict:
    cfg = resolving_view(server_args)

    # DCP（decode 上下文并行）分支：注意力后端、KV dtype、通信后端
    # 都有强约束，且与 FlashInfer 版本能力绑定。
    if cfg.dcp_size > 1:
        overrides = {}
        if cfg.enable_symm_mem:
            logger.warning(
                "Kimi-K3 DCP disables --enable-symm-mem due to decode CUDA "
                "graph correctness issues."
            )
            overrides["enable_symm_mem"] = False

        if cfg.speculative_algorithm == "DSPARK":
            # DSPARK 的 target-verify 必须跑在 decode 后端
            #（cutedsl_mla 实现了 DCP signature），否则 trtllm_mla 的
            # 基础 _run_decode_kernel 会因多余的 kwargs 抛 TypeError。
            overrides["speculative_attention_mode"] = "decode"

        prefill_backend, decode_backend = attention_backends_of(cfg)
        if decode_backend == "cutedsl_mla" or decode_backend is None:
            _require_kimi_k3_cutedsl_dcp_support()
            overrides.update(
                prefill_attention_backend="trtllm_mla",
                decode_attention_backend="cutedsl_mla",
            )
```

```
    elif decode_backend == "tokenspeed_mla":
        overrides.update(
            prefill_attention_backend="tokenspeed_mla",
            decode_attention_backend="tokenspeed_mla",
            kv_cache_dtype="fp8_e4m3",
        )
    else:
        raise AssertionError(
            f"Decode attention backend for Kimi-K3 DCP must be "
            f"'cutedsl_mla' or 'tokenspeed_mla', got {decode_backend!r}."
        )

    if cfg.dcp_replicate_q_proj is None:
        overrides["dcp_replicate_q_proj"] = True
    # 非 fabric 设备回落 a2a，MNNVL fabric 走 fi_a2a
    overrides["dcp_comm_backend"] = (
        "fi_a2a" if is_mnnvl_fabric_device() else "a2a"
    )
    return overrides

# 非 DCP：只有 SM100/SM103 才干预默认后端
if not (get_platform().is_sm100 and get_platform().device_sm in (100, 103)):
    return {}

backends_unset = is_attention_backend_not_set(cfg)
if cfg.speculative_algorithm != "DSPARK":
    if not backends_unset:
        return {}
    return {
        "decode_attention_backend": "trtllm_mla",
        "prefill_attention_backend": "trtllm_mla",
    }

# DSPARK 分支：q_len 决定 verify kernel 是否能跑在 decode 后端；
# checkpoint 自动推断发生在 overrides 之后，draft 默认 block 7。
q_len = cfg.speculative_num_draft_tokens or (
    cfg.speculative_dspark_block_size + 1
    if cfg.speculative_dspark_block_size is not None
    else 8
)
overrides = {}
if backends_unset:
    backend = "trtllm_mla"
    overrides["decode_attention_backend"] = backend
    overrides["prefill_attention_backend"] = "trtllm_mla"
else:
    # 显式 backend 优先，但 speculative_attention_mode 是独立旋钮，
    # 仍需要声明——否则 verify 留在 prefill 后端，host-side plan
    # 默认 flashinfer 会带来每步 D2H 同步。
```

```
        _, backend = attention_backends_of(cfg)
    if _dspark_verify_on_decode_backend(backend, q_len, cfg.kv_cache_dtype):
        overrides["speculative_attention_mode"] = "decode"
    return overrides
```

test/registered/unit/test_model_override_split.py

把 " 禁止字段重叠 " 这个非机械决策固化为测试，保证 import 顺序不再承载行为，是拆分质量的契约标尺。

```
class TestModelOverrideSplit(CustomTestCase):
    def test_no_field_is_declared_by_two_family_modules(self):
        # 找被两个以上家族模块认领的架构（合法：例如 Qwen3NextForCausalLM
        # 的注意力形状来自 qwen3_5、MoE runner 来自 qwen3_moe）
        contested = {
            arch: fns for arch, fns in _MODEL_OVERRIDE_FNS.items() if len(fns) > 1
        }
        self.assertTrue(contested, "the scan found no architecture with two claimants")
        for arch, fns in sorted(contested.items()):
            with self.subTest(architecture=arch):
                seen: dict[str, str] = {}
                for fn in fns:
                    for field in _returned_field_names(fn):
                        earlier = seen.get(field)
                        self.assertIsNone(
                            earlier,
                            f"{arch}: {fn.__module__}.{fn.__name__} and {earlier} "
                            f"both declare {field!r}, so which one wins depends "
                            f"on the order of the imports in "
                            f"arg_groups/model_overrides/__init__.py",
                        )
                        seen[field] = f"{fn.__module__}.{fn.__name__}"

    def test_the_import_list_names_every_family_module(self):
        # importing 即注册；某模块从 import 列表漏掉 = 该家族静默失效，
        # 而直接 import 该 provider 的测试不会发现。
        package = pathlib.Path(model_overrides.__file__).parent
        on_disk = {
            path.stem for path in package.glob("*.py") if path.stem != "__init__"
        }
        imported = {
            alias.name
            for node in ast.walk(ast.parse((package / "__init__.py").read_text()))
            if isinstance(node, ast.ImportFrom)
            and node.module == "sglang.srt.arg_groups.model_overrides"
            for alias in node.names
        }
        self.assertEqual(on_disk, imported)
```

评论区精华

本 PR 没有人工 review 评论（Codex Review 完成且无 findings），核心讨论集中在 PR body 的作者自述中：

> "isort alphabetised that list mid-split and flipped which module won for InternS2Mobius。The guard is nota pinned order。An overlap is a defect on its own: nobody owns the value, and it gets settled by whatever order the imports happen to be in."

这是本 PR 最重要的设计交锋：一个纯机械拆分引发的 import 顺序敏感问题，最终用 " 禁止重叠声明 " 而非 " 固定 import 顺序 " 来根治，使 isort 可以自由排序列表。

> "That is how a change to get_model_config()'s cache semantics went green through PR CI and broke two days later in the nightly: the axis it broke was not being looked at."

作者以此说明 test_model_source_paths.py 的动机——测试空白会让回归在 PR CI 阶段漏网，并强调新测试用 mutation 验证过有效性，而不是假设有效。

- 家族模块字段重叠 vs import 顺序语义 (design): 放弃 pinned order，改为禁止同一架构的两个模块声明同一字段；test_model_override_split.py 在重叠发生当天失败，import 列表可自由排序。
- model-source 测试空白导致的回归漏网 (testing): 新增 test_model_source_paths.py，并用三个 mutation 验证测试有效性（删 GGUF 声明失败 3 例、删 ModelScope 下载目录查找失败 1 例、删 model_weights 写入失败 1 例）。
- stacked 依赖 #37086 与 base 重检 (other): 作者要求先合并 #37086，并提示合并后需重新检查 base 与重跑 CI。

风险与影响

- 风险：
  1. 注册顺序与 import 副作用的耦合：虽然守卫测试消除了同架构字段重叠，但 _PREDICATE_OVERRIDE_FNS 仍按注册顺序应用、常量表先于 callables；未来新增 predicate 或常量注册仍可能重新引入顺序敏感，目前只有 test_model_override_split.py 覆盖精确架构注册，predicate 顺序无对应守卫。
  2. ServerArgs 公共契约变更：pre_capture_activation_reserve_mb、ssl_verify、modelexpress_url、modelexpress_transport、kv_event_block_size、_parsed_modelexpress_config、_support_mamba_cache_extra_buffer 等 record 成员被删除；out-of-tree 插件或外部调用方若直接调用这些方法会立即 break，PR 未提供 deprecated shim 或迁移清单。
  3. import 副作用注册的静默失效：model_overrides/__init__.py 的 import 列表一旦漏掉某家族模块，对应模型的配置声明会静默不生效；test_the_import_list_names_every_family_module 用 glob 对比磁盘与 import 列表可兜底，但测试本身位于 CPU unit 套件，若有人改测试套件选择器可能失去保护。
  4. 覆盖验证范围：62-shape probe 覆盖配置解析行为，但 26 个家族模块分散后，跨家族公共逻辑变更需要同时改多个文件；遗漏风险由 split 测试兜底，但不存在端到端模型启

动覆盖（PR 声明无 kernel/model-forward 变更，nightly 风险较低）。 - 影响：对用户无运行行为影响：62-shape resolution probe 在分支 tip 和每个 commit 上IDENTICAL，live registry 快照 55 架构 /61 注册顺序一致，所有注册单元测试 0 回归。对系统的影响集中在配置导入图与 ServerArgs 接口面：overrides.py 精简接近一半，record 成员减少 7 个，配置解析的关注点从单文件分散到按家族组织的模块。对团队而言，新增模型家族时不再需要编辑所有家族共用的 3043 行文件，文件冲突面和认知负载显著下降；同时 " 不得重叠声明 " 成为新增配置字段时必须遵守的契约，测试会在合并前拦截违规。 - 风险标记：配置解析核心路径，ServerArgs 契约变更，import 副作用注册，跨模块重构

关联脉络

- PR #37086 （前置 PR：为 platform facts 提供单一地址）：PR body 声明 stacked on #37086；拆分被其阻塞，family 模块移动 probe reads 需要 platform facts 单一地址。
- PR #36897 Decouple speculative draft capacity from runtime state: 同批 config/arg_groups 系列重构，同样修改 arg_groups/overrides.py 与 runtime_context.py，可共享注册与解析约束。
- PR #36991 [Diffusion] Add exact component precision overrides: 同一配置覆盖体系的能力扩展，围绕 server_args/arg_groups 的组件级配置演进脉络。