PR #30070 完整报告

sgl-project/sglang

[refactor] Add predicate-keyed registration; migrate the Step3p family (stack 8/15)

合并时间：2026-07-04 17:21

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30070

执行摘要

- 一句话：新增谓词键注册并迁移 Step3p 声明式覆盖
- 推荐动作：值得精读以理解声明式覆盖系统的扩展点。register_model_override_predicate 的设计（谓词 - 函数对）简洁地解决了遗留分支的匹配问题。测试用例清晰地锁定了行为，可作为类似迁移的参考。

功能与动机

当前参覆盖系统仅支持精确架构名称匹配（MODEL_OVERRIDES 和 register_model_override），但存在大量通过子串谓词（如 "Step3p5ForCausalLM" in model_arch）匹配的遗留分支。这些分支必须迁移到声明式系统以统一配置解析流程。本 PR 添加谓词键注册机制，为后续逐步迁移遗留分支铺平道路。

实现拆解

1. 在 overrides.py 中添加谓词键基础设施：新增全局列表 _PREDICATE_OVERRIDE_FNS 存储谓词 - 函数对；实现 register_model_override_predicate(predicate) 装饰器，将 (predicate, fn) 追加至该列表。
2. 抽取 _invoke_provider 帮助函数：将重复的类型检查与调用逻辑封装为 _invoke_provider(fn, server_args, hf_config)，在精确键和谓词键路径中共用。
3. 修改 collect_model_override_declarations：执行顺序变为常量 -> 精确键函数 -> 谓词键函数（按注册顺序）。谓词键函数仅当 predicate(architecture) 为 True 时执行。
4. 注册 Step3p 声明：使用 register_model_override_predicate 注册 _step3p_overrides，包含条件设置 enable_multi_layer_eagle、swa_full_tokens_ratio = 1.0 和 disable_hybrid_swa_memory = True（当 enable_hierarchical_cache 时）。
5. 标记 server_args.py 中的字段：将 swa_full_tokens_ratio 和 disable_hybrid_swa_memory 的 Arg 包装添加 model_overridable=True，并从 _handle_model_specific_adjustments 的 Step3p 分支中删除这些赋值（仅保留 attention_backend 自动选择）。
6. 更新 runtime_context.py 的 Flags：在 Flags 类中添加 swa_full_tokens_ratio: float = 0.8 和 disable_hybrid_swa_memory: bool = False 字段。
7. 测试覆盖：新增 test_predicate_keyed_provider 验证三层顺序正确性；新增 test_step3p_hierarchical_cache_golden 和 test_step3p_declarations_at_callable_level

锁定 Step3p 声明行为；更新白名单测试和 _IsolatedRegistry 的 setUp 以重置 _PREDICATE_OVERRIDE_FNS。

关键文件：

- python/sglang/srt/arg_groups/overrides.py（模块 覆盖系统；类别 source；类型 core-logic；符号 register_model_override_predicate, _invoke_provider, _step3p_overrides, _PREDICATE_OVERRIDE_FNS）：核心实现：添加谓词键注册机制和 Step3p 声明，修改集合顺序。
- test/registered/unit/test_model_overrides.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_predicate_keyed_provider, _exact, _by_predicate, test_step3p_hierarchical_cache_golden）：新增和修改测试以验证谓词键注册、白名单扩展和 Step3p 黄金数据。
- python/sglang/srt/server_args.py（模块 配置层；类别 source；类型 core-logic）：标记 SWA 字段为可模型覆盖，删除 Step3p 分支中的硬编码赋值。
- python/sglang/srt/runtime_context.py（模块 上下文；类别 source；类型 core-logic）：在 Flags 容器中添加对应的覆盖字段以便运行时读取。

关键符号：register_model_override_predicate, _invoke_provider, collect_model_override_declarations, _step3p_overrides, test_predicate_keyed_provider, test_step3p_hierarchical_cache_golden, test_step3p_declarations_at_callable_level

关键源码片段

python/sglang/srt/arg_groups/overrides.py

核心实现：添加谓词键注册机制和 Step3p 声明，修改集合顺序。

```
# 新增：谓词键提供者列表，按注册顺序存储 (predicate, fn) 对
_PREDICATE_OVERRIDE_FNS: List[Tuple[Callable[[str], bool], Callable[..., dict]]] = []


def register_model_override_predicate(predicate: Callable[[str], bool]):
    """注册一个通过架构名称谓词匹配的派生覆盖提供者。

    签名与 ``register_model_override`` 相同：装饰的函数接收
    ``(server_args, hf_config)`` 并返回 ``{field: value}`` 字典。
    """
    def decorator(fn: Callable[..., dict]) -> Callable[..., dict]:
        _PREDICATE_OVERRIDE_FNS.append((predicate, fn))
        return fn
    return decorator


def _invoke_provider(fn: Callable[..., dict], server_args: Any, hf_config: Any) -> Dict[str, Any]:
    """调用提供者并校验返回类型，减少重复。"""
    declared = fn(server_args, hf_config)
    if not isinstance(declared, dict):
        raise TypeError(
```

```
            f"model override provider {fn.__qualname__} must return a dict, "
            f"got {type(declared).__name__}"
        )
    return declared


def collect_model_override_declarations(
    architecture: str, server_args: Any, hf_config: Any
) -> List[Tuple[str, Dict[str, Any]]]:
    """收集常量->精确键->谓词键的声明列表，空声明被丢弃。"""
    declarations: List[Tuple[str, Dict[str, Any]]] = []
    const = MODEL_OVERRIDES.get(architecture)
    if const:
        declarations.append((f"MODEL_OVERRIDES[{architecture!r}]", dict(const)))
    for fn in _MODEL_OVERRIDE_FNS.get(architecture, ()):
        declared = _invoke_provider(fn, server_args, hf_config)
        if declared:
            declarations.append((fn.__qualname__, dict(declared)))
    for predicate, fn in _PREDICATE_OVERRIDE_FNS:
        if predicate(architecture):
            declared = _invoke_provider(fn, server_args, hf_config)
            if declared:
                declarations.append((fn.__qualname__, dict(declared)))
    return declarations


# 注册 Step3p 家族的覆盖（EAGLE 和分层缓存 SWA 配置）
@register_model_override_predicate(
    lambda arch: "Step3p5ForCausalLM" in arch or "Step3p7ForConditionalGeneration" in arch
)
def _step3p_overrides(server_args: Any, hf_config: Any) -> dict:
    result = {}
    if server_args.speculative_algorithm == "EAGLE":
        result["enable_multi_layer_eagle"] = True
        logger.info("Enable multi-layer EAGLE for Step3p5ForCausalLM.")
    if server_args.enable_hierarchical_cache:
        result["swa_full_tokens_ratio"] = 1.0
        result["disable_hybrid_swa_memory"] = True
        logger.info("Step3p5: reset swa_full_tokens_ratio to 1.0, disable hybrid SWA.")
    return result
```

test/registered/unit/test_model_overrides.py

新增和修改测试以验证谓词键注册、白名单扩展和 Step3p 黄金数据。

```
class TestModelOverrideRegistry(_IsolatedRegistry):
    # ... 已有测试 ...

    def test_predicate_keyed_provider(self):
        from sglang.srt.arg_groups.overrides import register_model_override_predicate
```

```
        @register_model_override("FakeStep9ForCausalLM")
        def _exact(server_args, hf_config):
            return {"a": 1}

        @register_model_override_predicate(lambda arch: "Step9" in arch)
        def _by_predicate(server_args, hf_config):
            return {"b": 2}

        # 匹配架构：精确键先，谓词键后
        self.assertEqual(
            collect_model_override_declarations("FakeStep9ForCausalLM", None, None),
            [(_exact.__qualname__, {"a": 1}), (_by_predicate.__qualname__, {"b": 2})],
        )
        # 不匹配架构：谓词不触发
        self.assertEqual(
            collect_model_override_declarations("OtherForCausalLM", None, None), []
        )


class TestGoldenModelOverrides(_IsolatedPublish):
    # ... 已有方法 ...

    def test_step3p_hierarchical_cache_golden(self):
        config_extra = {
            "layer_types": ["sliding_attention", "full_attention"],
            "sliding_window": 64,
        }
        sa = self._construct(
            "Step3p5ForCausalLM",
            "llama",
            config_extra=config_extra,
            enable_hierarchical_cache=True,
        )
        # 验证声明覆盖结果与遗留分支一致
        self.assertEqual(sa.swa_full_tokens_ratio, 1.0)
        self.assertTrue(sa.disable_hybrid_swa_memory)
```

评论区精华

无 review 评论。PR 由作者自行合并，推测属于成熟的重构栈，内部评审已在先前 PR 完成。

- 暂无高价值评论线程

风险与影响

- 风险：主要风险：谓词匹配可能因谓词过于宽泛而错误地触发在其他架构上（当前谓词明确限定为 Step3p5ForCausalLM/Step3p7ForConditionalGeneration，风险低）。三层顺序（const->exact->predicate）依赖测试锁定，若未来添加新谓词可能干扰顺序。已删除遗留分支中对 EAGLE 和 SWA 的赋值，若迁移不完全可能导致行为差异（但 golden 测试精确

验证了预期输出）。

- 影响：直接影响 Step3p 家族模型，其配置覆盖现在完全由声明式系统处理，行为应与迁移前一致。对其他模型无影响。对团队而言，明确了谓词键迁移模式，未来可将更多遗留分支以此方式迁移。字段 swa_full_tokens_ratio 和 disable_hybrid_swa_memory 变为可模型覆盖，增加了配置灵活性。
- 风险标记：核心路径变更，谓词匹配顺序依赖，遗留行为精确性

关联脉络

- PR #30071 [refactor] Sweep disable_hybrid_swa_memory writers; close the dtype family (stack 9/15): 同栈的后续 PR，继续迁移 disable_hybrid_swa_memory 和 dtype 字段的写入者。
- PR #30072 [refactor] Add the post-process resolution stage; migrate sampling_backend (stack 10/15): 同栈后续 PR，添加后处理解析阶段并迁移 sampling_backend。
- PR #30073 [refactor] Migrate the attention_backend resolution chain (stack 11/15): 同栈后续 PR，迁移 attention_backend 解析链。
- PR #30074 [refactor] Migrate the page_size resolution chain (stack 12/15): 同栈后续 PR，迁移 page_size 解析链。
- PR #30075 [refactor] Migrate the moe_runner_backend / quantization resolution chains (stack 13/15): 同栈后续 PR，迁移 MoE 后端和量化解析链。
- PR #30076 [refactor] Migrate the DeepSeek family and the parallel-request chains (stack 14/15): 同栈后续 PR，迁移 DeepSeek 家族和并行请求链。
- PR #30077 [refactor] Rename Arg.model_overridable to Arg.resolvable (stack 15/15): 同栈最终 PR，重命名 model_overridable 为 resolvable。