PR #34309 完整报告

sgl-project/sglang

[CI] Prune redundant CPU test overhead

合并时间：2026-08-14 10:51

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34309

执行摘要

- 一句话：静态 ratchet 迁至 pre-commit，CPU CI 耗时降约 8%
- 推荐动作：值得精读。三个设计决策尤其值得借鉴：(1) 把“测试自身的测试是否真的被执行”纳入 review 检查清单（本次发现 test_check_no_bare_pytest_main.py 是死代码）；(2) 跨文件缓存键一致性用可执行检查器固化，而不是靠人肉同步 build workflow 与 restore action；(3) 穷举覆盖到确定性覆盖的等价性论证方法（保留全部有序对、边界与正反用例）。对维护大型测试套件的团队，本 PR 的迁移节奏与 review 修复质量是很好的范本。

功能与动机

PR body 的核心诉求是去除 CPU 注册套件中的“冗余开销”：7 个 source/AST ratchet 以注册测试运行时，要为每次运行付出 subprocess 启动与 sglang.test.test_utils 导入成本，而它们本质是纯 AST/ 正则扫描；logprob 两个测试用穷举笛卡尔积（chunk-stitching 2,800 组合、fast-input 155 组合）换取大量冗余覆盖；cargo test --workspace 在每次 Base-a 门禁都跑 158s，即使 Rust 扩展缓存命中也照跑不误。PR 给出实测对照：聚合 CPU runner 时间 -413s（-7.8%），直接受影响负载缓存命中 -83%、未命中 -77%。

实现拆解

1. 静态 ratchet 迁出注册套件：7 个 AST/ 正则守卫从 test/registered/unit/（含 spec 子目录）迁至 scripts/lint/check_.py，移除 register_cpu_ci 与 CustomTestCase，改为纯函数 check_() + AssertionError；包根路径从 import sglang 改为相对 __file__ 计算，彻底摆脱 torch 与测试框架依赖。其中 check_global_config_read_ratchet.py 引入 @functools.cache 的 parsed_modules()，让字段读、configured*_size 调用点、import 重命名三个扫描器共享一次全包 AST 解析。对应删除 test/registered/unit/test_no_bare_pytest_main.py、test_server_args_mutation_ratchet.py、test_legacy_global_ratchet.py、test_module_state_ratchet.py、test_cargo_workspace.py 等 7 个旧测试文件。
2. pre-commit 接线与自测钩子：.pre-commit-config.yaml 新增 12 个 checker 条目（check_static_ratchets.py 作为聚合入口），并新增 check-lint-script-tests hook，用 unittest discover -s scripts/lint -p 'test_check_*.py' 跑 checker 自身单测，解决 review 指出的“测试文件永不运行”问题。
3. logprob 覆盖重构：新增 python/sglang/test/logprob_test_utils.py 的共享 coverage_cases 生成器，chunk-stitching 覆盖集从 2,800 组合降到 84、fast-input 从 155 降到 40，保留全部菜单项、有序对与正反 width-3/width-4 异构用例；菜单顺序成为

load-bearing，两个测试文件均加注释说明。

4. cargo workspace 测试挪位：删除 test/registered/rust/test_cargo_workspace.py，改为在 .github/workflows/_pr-test-rust-ext-build.yml 的缓存未命中路径执行 scripts/ci/run_rust_workspace_tests.sh，带 300s 超时、独立 CARGO_TARGET_DIR 测试目录避免污染 py3.12 指纹，测试脚本纳入 hashFiles 缓存键；新增 scripts/lint/check_rust_ext_cache_prefix.py 保证 build workflow 与 download action 两侧缓存前缀和 hash 输入一致。
5. 配套收尾：scripts/ci/check_registered_tests.py 迁至 scripts/lint/ 并集中注册测试过滤逻辑；.claude/skills/sglang-runtime-context/SKILL.md 与 .claude/rules/unit-test-admission.md 的 ratchet 路径指引同步更新；server_args.py 等源码注释中的旧测试文件指针修正。

关键文件：

- scripts/lint/check_global_config_read_ratchet.py（模块 静态检查；类别 source；类型 rename-or-move；符号 _field_reads, _parsed_modules, check_global_config_read_ratchet, check_configured_size_call_sites）：迁移与重构最重的 ratchet：从 unittest 类改为纯函数检查器，引入带缓存的 _parsed_modules() 让 3 个扫描器共享一次全包 AST 解析，是“去除注册测试子进程 / 导入开销”的代表作。
- scripts/lint/check_no_bare_pytest_main.py（模块 静态检查；类别 source；类型 entrypoint；符号 is_main_guard, is_pytest_main_call, is_exit_call, assigned_names）：新增检查器，把旧的“裸 Expr 匹配”升级为带父节点表与退出码传播分析的 AST 精确匹配（sys.exit / raise SystemExit / 赋值后 exit 均合规），是 review 重点打磨对象，并配套 11 个单测。
- .pre-commit-config.yaml（模块 前置钩子；类别 config；类型 configuration）：整个迁移的接线中枢：新增 12 个 checker 条目 + check-lint-script-tests 单测发现 hook，决定静态守卫在开发者侧的强制时机；review 中“测试文件永不运行”的发现由这里修复。
- .github/workflows/_pr-test-rust-ext-build.yml（模块 CI 编排；类别 infra；类型 core-logic）：cargo test --workspace 从 Base-a 移除后在这里的缓存未命中路径执行；配合独立 CARGO_TARGET_DIR 与 300s 超时，是门禁 158s 开销归零的关键载体，也是缓存键一致性检查的对象。
- scripts/lint/check_decode_bookkeeping_ownership.py（模块 静态检查；类别 source；类型 rename-or-move；符号 check_bookkeeping_sites_match_owner_allowlist, check_spec_v2_draft_workers_do_no_scheduler_bookkeeping, draft_worker_classes, _scan_class_subtree）：从 test/registered/unit/spec/ 迁入的 AST 守卫，保护调度簿记时钟（decode_batch_idx / kv_committed_len 等）不被 spec-v2 draft worker 重复推进；迁移后改为两个 check* 函数并在 __main__中直接执行。
- scripts/lint/check_rust_ext_cache_prefix.py（模块 静态检查；类别 source；类型 core-logic；符号 hashed_inputs, main）：新增长效守卫：解析 build workflow 与 download action 的 YAML，确保两侧缓存前缀与 hashFiles 输入一致，防止缓存静默失效回退源码编译；直接对应 PR 的缓存键同步改动。
- scripts/lint/check_parallel_adoption_ratchet.py（模块 静态检查；类别 source；类型 rename-or-move；符号 check_parallel_adoption_ratchet, _BANNED_CALLS, _EXEMPT）

：从 test/registered/unit/ 迁入的并行拓扑 getter 守卫，禁止 models/layers 目录直呼 legacy parallel_state getter；迁移后由 unittest 类收敛为单个 check 函数，_SRT_ROOT 改为相对路径推导。

- python/sglang/test/logprob_test_utils.py（模块 测试工具；类别 test；类型 test-coverage；符号 coverage_cases）：新增共享 coverage_cases 生成器，修复 review 指出的两处逐字重复；确定性覆盖（84/40 例）取代穷举笛卡尔积（2,800/155 例），menu 顺序标记为 load-bearing。
- test/registered/unit/layers/test_logprob_chunk_stitching.py（模块 覆盖用例；类别 test；类型 test-coverage）：覆盖集从 2,800 组合降到 84 的载体，同时保留全部菜单项、有序对与正反 width-3/4 异构用例；展示"确定性边界覆盖替代穷举"的测试瘦身方法。
- scripts/ci/run_rust_workspace_tests.sh（模块 CI 脚本；类别 infra；类型 entrypoint）：承载 cargo test --workspace 的新脚本，300s 超时与独立 CARGO_TARGET_DIR 防挂起、防指纹污染；被 3 处 hashFiles 缓存键引用，因此保留在 scripts/ci/ 而非 scripts/lint/。

关键符号：check_global_config_read_ratchet, check_configured_size_call_sites, check_no_renamed_accessor_imports, find_bare_pytest_main, is_main_guard, is_pytest_main_call, propagates_exit_code, check_bookkeeping_sites_match_owner_allowlist, check_spec_v2_draft_workers_do_no_scheduler_bookkeeping, check_parallel_adoption_ratchet, check_rust_ext_cache_prefix, hashed_inputs, check_server_args_mutation_ratchet, check_legacy_global_ratchet, check_module_state_ratchet, coverage_cases

关键源码片段

scripts/lint/check_global_config_read_ratchet.py

迁移与重构最重的 ratchet：从 unittest 类改为纯函数检查器，引入带缓存的 _parsed_modules() 让 3 个扫描器共享一次全包 AST 解析，是"去除注册测试子进程 / 导入开销"的代表作。

```
# scripts/lint/check_global_config_read_ratchet.py
# 迁移自 test/registered/unit/test_global_config_read_ratchet.py:
# 去掉 register_cpu_ci / CustomTestCase / import sglang 后，
# 不再向注册测试的子进程与测试框架付费，规则与诊断完全保留。

import ast
from functools import cache
from pathlib import Path

# 用脚本位置推导包根，而不是 import sglang——后者会拉起 torch 等重依赖，
# 是"dependency-light"迁移的关键
_PACKAGE_ROOT = Path(__file__).resolve().parents[2] / "python" / "sglang"

# 槽位所有者：runtime_context 发布对象并提供命名访问器；
# server_args / arg_groups 是解析管线本身，扫描时整段跳过
_SLOT_OWNERS = ("srt/runtime_context.py", "srt/server_args.py", "srt/arg_groups/")

@cache
```

```
def _parsed_modules():
    """全包 AST 解析缓存：字段读、configured_*_size 调用点、
    导入重命名 3 个扫描器共享同一份解析结果。"""
    modules = []
    for path in sorted(_PACKAGE_ROOT.rglob("*.py")):
        try:
            tree = ast.parse(path.read_text())
        except SyntaxError:
            continue # 语法错误文件跳过，与旧测试行为一致
        modules.append((path.relative_to(_PACKAGE_ROOT).as_posix(), tree))
    return modules


def _field_reads():
    """业务代码对进程级 ServerArgs 字段的直接读与别名读计数。"""
    direct, alias = [], []
    for rel, tree in _parsed_modules():
        if rel.startswith(_SLOT_OWNERS):
            continue
        inert = frozenset(
            name for path_, name in _INERT_DYNAMIC_READS if path_ == rel
        )
        module_direct, module_alias = _collect(rel, tree, inert)
        direct += module_direct
        alias += module_alias
    return direct, alias


def _check_count(kind, reads, baseline):
    # 只允许下降：超过基线说明业务代码又直接读了发布记录；
    # 少于基线则要求把基线下调，把进展钉死
    if len(reads) > baseline:
        raise AssertionError(
            f"{kind} process-global config field reads grew: {len(reads)} > "
            f"baseline {baseline}. Read the namespace accessor for the "
            "field's namespace, or the owning runner for a per-runner "
            "field:\n" + "\n".join(reads)
        )


def check_global_config_read_ratchet():
    direct, alias = _field_reads()
    _check_count("direct", direct, _DIRECT_BASELINE)
    _check_count("alias-form", alias, _ALIAS_BASELINE)


if __name__ == "__main__":
    # 文件内还定义了 check_configured_size_call_sites() 与
    # check_no_renamed_accessor_imports()，连同本函数一并被执行
```

```
    check_global_config_read_ratchet()
    check_configured_size_call_sites()
    check_no_renamed_accessor_imports()
```

scripts/lint/check_no_bare_pytest_main.py

新增检查器，把旧的“裸 Expr 匹配”升级为带父节点表与退出码传播分析的 AST 精确匹配（sys.exit / raise SystemExit / 赋值后 exit 均合规），是 review 重点打磨对象，并配套 11 个单测。

```
# scripts/lint/check_no_bare_pytest_main.py（全新 check_*）
# 目标：禁止 `__main__` 块里丢弃 pytest.main() 返回值——注册测试以子进程方式
# 跑时，退出码不传播会让 CI 把失败当成功。旧版只匹配“裸 Expr 语句”，
# 新版用父节点表追查赋值与 exit 传播，误报 / 漏报都更少。

def is_main_guard(node):
    """匹配 `__name__ == "__main__"`（左右两侧皆可）。"""
    if not isinstance(node, ast.Compare) or len(node.ops) != 1:
        return False
    if not isinstance(node.ops[0], ast.Eq):
        return False
    sides = [node.left, *node.comparators]
    has_name = any(
        isinstance(side, ast.Name) and side.id == "__name__" for side in sides
    )
    has_main = any(
        isinstance(side, ast.Constant) and side.value == "__main__" for side in sides
    )
    return has_name and has_main


def is_pytest_main_call(node):
    """匹配 pytest.main(...) 调用（容忍 `pytest . main` 这类空白写法）。"""
    if not isinstance(node, ast.Call):
        return False
    func = node.func
    return (
        isinstance(func, ast.Attribute)
        and func.attr == "main"
        and isinstance(func.value, ast.Name)
        and func.value.id == "pytest"
    )


def is_exit_call(node, parents):
    """sys.exit(...)，或真正被 raise 的 SystemExit(...)。"""
    if not isinstance(node, ast.Call):
        return False
    func = node.func
    if (
```

```
            isinstance(func, ast.Attribute)
            and func.attr == "exit"
            and isinstance(func.value, ast.Name)
            and func.value.id == "sys"
        ):
            return True
        parent = parents.get(id(node))
        return (
            isinstance(func, ast.Name)
            and func.id == "SystemExit"
            and isinstance(parent, ast.Raise)
            and parent.exc is node
        )


    def find_bare_pytest_main(path):
        """返回第一个“裸 pytest.main()”的行号；没有则返回 None。
        辅助函数 runtime_nodes / exited_names / propagates_exit_code 定义在同文件。"""
        try:
            source = path.read_text(encoding="utf-8")
        except (OSError, UnicodeDecodeError):
            return None
        # 快速路径：没有 __main__ 或 pytest.main 就不做 AST 解析
        if "__main__" not in source or _PYTEST_MAIN.search(source) is None:
            return None
        try:
            tree = ast.parse(source, filename=str(path))
        except SyntaxError:
            return None

        for node in ast.walk(tree):
            if not isinstance(node, ast.If) or not is_main_guard(node.test):
                continue
            # 把整个 if 体（含嵌套语句）展平，调用与 sys.exit() 往往分属两条语句
            nodes = [n for statement in node.body for n in runtime_nodes(statement)]
            parents = {
                id(child): parent
                for parent in nodes
                for child in ast.iter_child_nodes(parent)
            }
            exited = exited_names(nodes, parents) # 先收集被 exit 的名字
            for candidate in nodes:
                if is_pytest_main_call(candidate) and not propagates_exit_code(
                    candidate, parents, exited
                ):
                    return candidate.lineno
        return None
```

scripts/lint/check_rust_ext_cache_prefix.py

新增长效守卫：解析 build workflow 与 download action 的 YAML，确保两侧缓存前缀与 hashFiles 输入一致，防止缓存静默失效回退源码编译；直接对应 PR 的缓存键同步改动。

```
# scripts/lint/check_rust_ext_cache_prefix.py
# 背景：Rust 扩展缓存的"写入前缀 + hash 输入"定义在 build workflow 里，
# 恢复动作在 download action 里，两文件互不引用；任一侧不一致都会让
# 所有 worker pool 在安装期静默回退到源码编译，且 CI 不会报错。

import re
import sys
import yaml

_BUILD_WORKFLOW = ".github/workflows/_pr-test-rust-ext-build.yml"
_DOWNLOAD_ACTION = ".github/actions/download-rust-ext/action.yml"

# 提取 hashFiles(...) 调用与其中的单引号参数
_HASH_FILES = re.compile(r"hashFiles\(([^)]*)\)")
_QUOTED = re.compile(r"'([^']*)'")


def hashed_inputs(path):
    """取出文件里每个 hashFiles(...) 的参数元组。"""
    with open(path, encoding="utf-8") as f:
        text = f.read()
    return [tuple(_QUOTED.findall(args)) for args in _HASH_FILES.findall(text)]


def main():
    with open(_BUILD_WORKFLOW, encoding="utf-8") as f:
        workflow = yaml.safe_load(f)
    with open(_DOWNLOAD_ACTION, encoding="utf-8") as f:
        action = yaml.safe_load(f)

    # yaml 1.1 会把 `on:` 键解析成布尔 True，所以两个键都要兜底
    triggers = workflow.get("on", workflow.get(True))
    save_prefix = triggers["workflow_call"]["inputs"]["cache_key_prefix"]["default"]
    restore_prefix = action["inputs"]["cache_key_prefix"]["default"]

    if save_prefix != restore_prefix:
        print("ERROR: rust-ext cache_key_prefix defaults do not match.")
        print(f"  {_BUILD_WORKFLOW} saves under:    {save_prefix}")
        print(f"  {_DOWNLOAD_ACTION} restores with: {restore_prefix}")
        return 1

    # 任一侧单独增删 hash 输入都会让另一侧永久 miss
    sites = [(_BUILD_WORKFLOW, i) for i in hashed_inputs(_BUILD_WORKFLOW)]
    sites += [(_DOWNLOAD_ACTION, i) for i in hashed_inputs(_DOWNLOAD_ACTION)]

    if not sites:
```

```
            print("ERROR: no hashFiles(...) cache key found; this check is dead.")
            return 1
        if len({inputs for _, inputs in sites}) > 1:
            print("ERROR: rust-ext cache key inputs do not match.")
            for path, inputs in sites:
                print(f"  {path}: {list(inputs)}")
            return 1
        return 0


    if __name__ == "__main__":
        sys.exit(main())
```

评论区精华

合并人 hnyls2002 对整体工作给出肯定：“Nice win, and the cache-key work is careful -- all three hashFiles sites moved together and the restore job's sparse-checkout picked up the new script, which is the easy thing to miss.” 六条内联意见全部指向实质问题：目录归属（check_*.py 应从 scripts/ci/ 迁到 scripts/lint/）、checker 单测零引用（死代码）、cargo 测试 deactivate 后跑引发的解释器指纹污染、缺失 300s 超时、文档指针未更新、logprob 生成器重复且 menu 顺序成为隐性约束。

- lint 脚本应放 scripts/lint/ 而非 scripts/ci/ (design): 已按建议迁移：12 个 checker 及其测试全部移入 scripts/lint/，Rust 工作流脚本保留在 scripts/ci/。
- checker 自身的单元测试是死代码 (testing): 合并前新增 pre-commit hook（python3 -m unittest discover -s scripts/lint -p 'test_check_*.py'）并纳入 lint 检查触发范围。
- cargo 测试在 deactivate 后运行可能污染 py3.12 目标指纹 (correctness): 已把 CARGO_TARGET_DIR 指向独立 test 目录（cargo_target_root/test），并将调用移入与 build 相同的解释器上下文。
- cargo 测试缺失 300s 超时 (correctness): 已在 run_rust_workspace_tests.sh 中恢复 timeout 300 cargo test --workspace。
- server_args.py 注释里的 ratchet 文件路径没跟着迁 (documentation): 已同步更新 runtime-context 技能文档与 unit-test-admission 规则，并修正源码内注释指针。
- logprob 覆盖生成器两处重复、menu 顺序成为隐性约束 (design): 已抽出 python/sglang/test/logprob_test_utils.py 的共享 coverage_cases 生成器，并在两处标注 menu 顺序为 load-bearing。

风险与影响

- 风险：
  1. 守卫执行时机变化：静态守卫从“注册 CPU 套件必然运行”变为“pre-commit + lint CI 运行”，依赖开发者本地钩子与 CI lint job 双保险；若有绕过 pre-commit 的提交且 lint 未覆盖，保护会空转。
  2. logprob 覆盖从 2,800/155 组合锐减到 84/40：确定性集合保留了全部菜单项、有序对与正反 width-3/4 用例，但穷举特有的畸形组合探测能力不再有，依赖特定组合才触发的

回归需要靠真实模型测试兜底。

3. cargo test 仅在缓存未命中路径执行：缓存命中时 Rust 新增测试的回归风险缺少直接承载，300s 超时与独立 CARGO_TARGET_DIR 已缓解挂起与指纹污染问题。
4. check_rust_ext_cache_prefix.py 基于 yaml.safe_load 与正则解析 hashFiles 参数，若 workflow 语法变化（如改 ${{ }} 拼接表达式）可能失效或误报；代码已对 yaml 1.1 把 on 键解析为 True 做了兜底。
5. 78 个文件搬迁、近千行删除对历史追溯有影响，review 时代码注释与文档中的路径指针曾遗漏更新（已修复）。 - 影响：对用户与推理运行时无影响：除注释外 python/sglang/srt 业务代码未动。对工程效率影响显著：CPU CI 聚合时间 -7.8%，Base-a 门禁每跑约 -58s，直接受影响负载缓存命中 -83%、未命中 -77%；开发者本地获得 12 个可离线运行的静态守卫与 11 个 checker 单测。对团队而言，确立了"静态契约走 pre-commit、注册测试专注运行时行为"的 CI 分层范式，scripts/lint/ 成为新的守卫目录，后续新增 ratchet 有明确归属。 - 风险标记：静态守卫依赖 pre-commit 执行时机，logprob 覆盖组合大幅缩减，cargo 测试仅缓存未命中路径运行，78 文件大规模搬迁，缓存键检查依赖 yaml 解析与正则

关联脉络

- PR #34730 [Core] Organize environment variable registry: 同属"运行时上下文 / 配置解析基础设施收口"方向：本 PR 迁移的多个 ratchet（server_args 读、legacy global、parallel 采用、module state）正是该方向演进的静态守卫，把配置访问约束前置到 pre-commit。