PR #36241 完整报告

sgl-project/sglang

[CI] Cut repeated tokenizer loads, serial subprocesses and a double scan

合并时间：2026-08-25 09:22

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36241

执行摘要

- 一句话：削减三个 CPU 测试的重复加载与串行子进程开销
- 推荐动作：值得精读。该 PR 展示了如何系统性地诊断 CI 性能瓶颈（逐个文件计时、区分根因、校准基线），并给出三个可复用的优化范式：lru_cache 消除重复对象构建、ThreadPoolExecutor 并行化独立子进程、字符串预过滤减少 AST 解析面。作者对每个优化都提供了实测数据、等价性论证和权衡说明，尤其对预过滤弱点和 -k 行为变化的坦诚披露，是写高质量 CI 优化 PR 的范例。关注 test_import_surface 预过滤后续是否被替换为更强方案，以及 base-c 测试是否跟进 tokenizer 共享。

功能与动机

PR body 明确指出这是对 shard-7 超时（run 32786070189）的跟进：对所有 CPU 测试逐一计时后发现三个昂贵文件，且每个的根因都不同，均可在不动断言的前提下修复。其中 test_function_call_parser.py 每次运行会构建 15 次同一个 DeepSeek-V3.2 tokenizer，约 30s 阻塞在网络往返上，既是性能问题也是 flakiness 来源；test_benchmark_datasets_api.py 的 21.45s 测试时间被 5 个串行 CLI 子进程吃掉；test_import_surface.py 则对 5,479 个 py 文件重复 ast.parse 两遍。

实现拆解

实现分三个独立优化，分别对应一个测试文件：

1. test_function_call_parser.py — 共享 tokenizer：新增模块级 @functools.lru_cache(maxsize=None) 包装的 _shared_tokenizer(path)，其中延迟导入 get_tokenizer，将 TestDeepSeekV32Detector.setUp 和 TestDeepSeekV4Detector.setUp 中的直接 get_tokenizer("deepseek-ai/DeepSeek-V3.2") 替换为 _shared_tokenizer(...) 调用。因 15 个测试里只有 6 个用到 self.tokenizer 且仅做只读的 .encode() / .decode()，共享安全。优化后该文件从 45.92s 降至 3.14s，并消除了 15 次 HF hub 往返。
2. test_benchmark_datasets_api.py — 并发运行独立 CLI 子进程：将五个 python -m sglang.benchmark.serving 的独立调用抽取为 _BENCH_SERVING_CLI_CASES 字典（help、invalid_distribution、flush_cache_timeout、zipf_without_alpha、uniform_with_alpha），新增 @functools.lru_cache(maxsize=1) 的 _bench_serving_cli_results()，内部用 ThreadPoolExecutor(max_workers=len(cases)) 并发提交，返回各 case 的 CompletedProcess。原测试中的 subprocess.run 调用改为从

结果字典取值，断言不变。实测串行 23.72s → 并行 5.21s，stdout 字节级一致，峰值子进程 RSS 从 635MB 升至 ~3.2GB，仍在 runner 可承受范围。

3. test_import_surface.py — 合并一次扫描并加预过滤：将原来两个测试各自独立遍历全部 .py 文件并 ast.parse 的逻辑，合并进新的 @functools.lru_cache(maxsize=None) 的 _scan_root(root)，一次遍历同时收集 facade 未导出符号和深层导入违规项，并增加 if PACKAGE not in source 字符串预过滤，跳过绝大多数无关文件。原两次全量解析共 15.16s（解析 5,478 个文件），优化后单次 0.45s（仅解析 65 个文件）。等价性通过 diff 原始命中集合验证：280 个 facade 导入名、8 个深层导入命中完全一致。
4. 配套 est_time 修正：三个文件在 register_cpu_ci 中的预估耗时分别更新为 25（import_surface）、30（benchmark_datasets_api）、20（function_call_parser），使分片均衡器不再被低估误导。base-c-test-cpu 注册的 46/70 秒无 base-c 实测数据，未改动。另外 PR body 说明 test_dflash_logits.py 虽被调查但刻意不动，只建议把 est_time 从 1 改成 35，单独提交。

关键文件：

- test/registered/bench_fn/test_benchmark_datasets_api.py（模块 基准数据集；类别 test；类型 test-coverage；符号 _bench_serving_cli_results）：将 5 个串行 CLI 子进程改为并发执行，通过 lru_cache 缓存结果，耗时从 26.88s 降至 10.96s，是三个优化中收益最大且涉及并发资源权衡的文件。
- test/registered/kernels/ops/diffusion/test_import_surface.py（模块 导入面守卫；类别 test；类型 test-coverage；符号 _scan_root）：将两个测试各自的全仓库 AST 扫描合并为单个 lru_cache 扫描，并新增包名字符串预过滤，使解析文件数从 5,478 降至 65，耗时从 15.16s 降至 0.45s。
- test/registered/unit/function_call/test_function_call_parser.py（模块 函数调用解析；类别 test；类型 test-coverage；符号 _shared_tokenizer）：消除同一个 DeepSeek-V3.2 tokenizer 在 setUp 中被重复构建 15 次的问题，引入 lru_cache 共享，文件耗时从 45.92s 降至 3.14s，同时移除每轮 CI 的 15 次 HF hub 网络往返。

关键符号：_shared_tokenizer, _bench_serving_cli_results, _scan_root

关键源码片段

test/registered/bench_fn/test_benchmark_datasets_api.py

将 5 个串行 CLI 子进程改为并发执行，通过 lru_cache 缓存结果，耗时从 26.88s 降至 10.96s，是三个优化中收益最大且涉及并发资源权衡的文件。

```
# 所有 CLI 用例集中定义，键名与测试中的取值一一对应
_BENCH_SERVING_CLI_CASES = {
    "help": ["--help"],
    "invalid_distribution": [
        "--dataset-name",
        "generated-shared-prefix",
        "--gsp-group-distribution",
        "invalid_name",
    ],
```

```
    "flush_cache_timeout": ["--flush-cache-timeout", "inf"],
    "zipf_without_alpha": [
        "--dataset-name",
        "generated-shared-prefix",
        "--gsp-group-distribution",
        "zipf",
        "--ready-check-timeout-sec",
        "0",
    ],
    "uniform_with_alpha": [
        "--dataset-name",
        "generated-shared-prefix",
        "--gsp-group-distribution",
        "uniform",
        "--gsp-zipf-alpha",
        "1.0",
        "--ready-check-timeout-sec",
        "0",
    ],
}


@functools.lru_cache(maxsize=1)
def _bench_serving_cli_results():
    # 并发拉起所有独立 CLI 子进程，互不影响断言结果；
    # lru_cache 保证整个测试进程内每个 case 只真正执行一次
    def run(args):
        return subprocess.run(
            [sys.executable, "-m", "sglang.benchmark.serving", *args],
            capture_output=True,
            text=True,
            timeout=180, # 兜底防挂死
        )

    with ThreadPoolExecutor(max_workers=len(_BENCH_SERVING_CLI_CASES)) as pool:
        futures = {
            name: pool.submit(run, args)
            for name, args in _BENCH_SERVING_CLI_CASES.items()
        }
    # 收集时按名称顺序取结果，保证确定性
    return {name: future.result() for name, future in futures.items()}
```

test/registered/kernels/ops/diffusion/test_import_surface.py

将两个测试各自的全仓库 AST 扫描合并为单个 lru_cache 扫描，并新增包名字符串预过滤，使解析文件数从 5,478 降至 65，耗时从 15.16s 降至 0.45s。

```
@functools.lru_cache(maxsize=None)
def _scan_root(root: str) -> tuple[frozenset[str], tuple[str, ...]]:
    """对指定根目录做一次遍历，返回 (未导出的 facade 导入名，深层导入违规位置)。
```

```
lru_cache 让两个测试共享同一次扫描，避免重复解析全仓库。
"""
unexported: set[str] = set()
offenders: list[str] = []
root_dir = _REPO_ROOT / root
if not root_dir.exists():
    return frozenset(), ()

for path in root_dir.rglob("*.py"):
    rel = path.relative_to(_REPO_ROOT).as_posix()
    # 包内部文件不在此守卫范围内
    if rel.startswith("python/sglang/kernels/ops/diffusion/"):
        continue
    try:
        source = path.read_text(encoding="utf-8")
    except UnicodeDecodeError:
        continue
    # 字符串预过滤：绝大多数文件根本不提及该包，跳过 ast.parse
    if PACKAGE not in source:
        continue
    try:
        tree = ast.parse(source)
    except SyntaxError:
        continue
    allowlisted = rel in _DEEP_IMPORT_ALLOWLIST
    for node in ast.walk(tree):
        if isinstance(node, ast.ImportFrom):
            # 从 facade 导入但符号未在 _EXPORTS 中 → 未导出
            if node.module == PACKAGE:
                unexported.update(
                    a.name
                    for a in node.names
                    if a.name not in _EXPORTS and not a.name.startswith("_")
                )
            # 非白名单文件直接导入 facade 子模块 → 深层导入
            elif (
                not allowlisted
                and node.module
                and node.module.startswith(f"{PACKAGE}.")
            ):
                offenders.append(f"{rel}:{node.lineno} imports {node.module}")
        elif isinstance(node, ast.Import) and not allowlisted:
            offenders.extend(
                f"{rel}:{node.lineno} imports {a.name}"
                for a in node.names
                if a.name.startswith(f"{PACKAGE}.")
            )
return frozenset(unexported), tuple(offenders)
```

test/registered/unit/function_call/test_function_call_parser.py

消除同一个 DeepSeek-V3.2 tokenizer 在 setUp 中被重复构建 15 次的问题，引入 lru_cache 共享，文件耗时从 45.92s 降至 3.14s，同时移除每轮 CI 的 15 次 HF hub 网络往返。

```
@functools.lru_cache(maxsize=None)
def _shared_tokenizer(path: str):
    # 延迟导入 get_tokenizer，避免加载本文件时引入额外依赖；
    # lru_cache 使得同一个模型路径在一轮测试中只构建一次 tokenizer，
    # 原先每个 setUp 都会重新走 HF hub 加载，15 个测试就是 15 次网络往返
    from sglang.srt.utils.hf_transformers_utils import get_tokenizer

    return get_tokenizer(path)
```

评论区精华

该 PR 没有正式的 review 评论（review_comments_count=0），但 PR body 中作者主动提出了两个判断点供审阅者权衡：

1. test_import_surface 的预过滤依赖较弱论证：from a . b import c 这种点号周围带空格的合法 Python 写法会绕过字符串子串预过滤。作者指出 black 会规范化这种写法、当前仓库不存在此类文件，但这是“当下为真”而非“构造上为真”。他给出替代方案：只保留共享缓存扫描、去掉预过滤，仍能把 15.16s 降到约 8.5s，且无此隐患。
2. test_benchmark_datasets_api 的 -k 行为变化：改动后，用 -k 单独运行某个 CLI 测试时会触发全部五个子进程（而不是原来只跑一个），这是以并发换取的代价。

另外，PR 提交后三次请求 /rerun-test，前两次因 main 有新提交导致分支 diverged 而未触发，rebase 后最终两个 workflow run 均通过（🤔），三个测试文件全部成功。

- 预过滤的语法覆盖缺口 (design): 作者保留了预过滤以换取最大加速（0.45s vs 8.5s），但明确给出替代方案：只保留共享缓存扫描、去掉预过滤，仍能减半耗时且无此隐患，留给审阅者权衡。
- -k 单测触发全部 CLI 子进程 (design): 作者明示该行为变化，接受此代价换取整体并行收益，未引发进一步讨论。
- CI rerun 与 rebase 要求 (other): 最终验证通过，三个测试文件均 OK。

风险与影响

- 风险：
  1. tokenizer 共享的并发与状态风险：_shared_tokenizer 用 lru_cache 共享同一个 tokenizer 对象，虽然测试里只调用 .encode() / .decode() 等只读方法，但若未来某个测试对 tokenizer 做有状态修改（如添加特殊 token），会污染其他测试。当前安全，但缺少显式文档约束。
  2. 并行子进程的资源风险：_bench_serving_cli_results 并发拉起 5 个 sglang.benchmark.serving 子进程，峰值 RSS 从 635MB 升至约 3.2GB。若 CI runner 内存配置变化或用例增多，可能 OOM；timeout=180 是兜底，但并发超时会同

时超时。

3. 预过滤漏检风险：test_import_surface.py 的 if PACKAGE not in source 子串预过滤，对 from sglang . kernels . ops . diffusion import x 这类带空白的合法写法会漏检（PR body 已明确说明）。当前仓库无此类代码，但这是"当下为真"而非结构保证，未来新增代码若绕开 black 格式化可能逃过守卫。
4. lru_cache 跨进程 / 跨运行语义：lru_cache 仅在单进程内生效。CI 中每个测试文件是独立 python3 <file> 子进程，所以缓存只影响文件内部重复调用；若未来 pytest 以多进程方式运行，缓存会被复制，不影响正确性。
5. est_time 调整的连锁影响：est_time 是分片均衡依据，更新后的数值直接影响 CI 分片布局。若估算偏差过大（比如低估或高估），可能引起新的分片不均衡。 - 影响：影响面集中在 CI 测试基础设施，不涉及任何运行时源码，对用户无直接功能影响。三个文件合计从约 94s（45.92 + 21.12 + 26.88）降至约 22s（3.14 + 7.61 + 10.96），直接改善 CPU CI 分片耗时与稳定性。同时每轮 CI 减少 15 次 HF hub 网络往返，降低外部网络抖动导致的 flakiness。团队收益是更快的 CI 反馈和更可靠的分片。由于 est_time 被修正，后续 shard 划分更准确，避免同类超时复发。 - 风险标记：CI 基础设施变更，并发子进程内存峰值上升，预过滤存在语法覆盖缺口，缺少 base-c 实测数据校准

关联脉络

- PR #36235 [CI] Stop the resolution ratchets re-parsing the whole package: PR body 明确提到本 PR 是 shard-7 超时调查的一部分，而该 PR 处理的是同一批昂贵 CPU 测试中的 package-AST-scan 模式，与本 PR 的 test_import_surface 优化属于同一轮 CI 性能治理，目标文件同为 test/registered/unit/server_args/test_resolution_is_reproducible.py 之外的其他扫描测试。
- PR #36242 [CI] Register test_dflash_logits at its real cost: PR body 专门说明 test_dflash_logits.py 被调查但刻意不改代码，只建议将 est_time 从 1 修正为 35，并称单独提交以便独立合并，与本 PR 同为 CI est_time 校准系列。
- PR #35929 Report the whole server's world size in the scheduler's internal state: 同一时期对 CI 测试基础设施的持续优化，涉及 test/registered/unit/managers 下的 est_time 与 shard 均衡，反映了仓库对 CI 分片超时的系统治理趋势。