PR #34520 完整报告

sgl-project/sglang

[Benchmark] Remove 22 unmaintained benchmarks

合并时间：2026-08-12 13:24

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34520

执行摘要

- 一句话：删除 22 个无人引用的过期 benchmark 目录
- 推荐动作：值得浏览 PR body 中的清理标准（无引用、无维护、第三方依赖、纯历史），可作为 benchmark 资产治理的参考。具体删除内容无需精读；若未来计划恢复某个 benchmark（如 json_schema 或 reasoning），可直接从该 PR 的 Git 历史提取。建议关注第三 commit 对 test_utils 死代码的清理范围。

功能与动机

PR body 给出的清理标准是：22 个目录在仓库中 unreferenced anywhere（无 CI job、test、doc 或 source comment 引用），数月到数年没有任何针对性改动，最近的触碰都是全仓库范围的 sweep；多个脚本 import guidance、lmql、dspy，均不是 sglang 依赖，导致部分 benchmark 已完全无法运行；benchmark_vllm_060 复现的是 SGLang v0.3.0 对比，属于纯历史资产。删除这些目录能降低仓库维护噪音，避免用户误用过期脚本。

实现拆解

1. 盘点与筛选：按 PR body 中的 4 条标准（无仓库内引用、长期无针对性改动、依赖非 sglang 第三方库、纯历史）筛出 22 个目录。
2. 删除主基准脚本：每个目录下通常包含 bench_sglang.py（SGLang DSL 版本）与 bench_other.py（guidance/lmql/dspy 对比版本），本次一并删除；同时删除配套辅助模块，如 benchmark/json_schema/bench_sglang.py 的 schema_gen/bench_schema 链路、benchmark/generative_agents/agent_functions.py 的智能体 prompt 库、benchmark/reasoning_benchmark 下的 answer_extraction.py 与 eval_utils.py。
3. 清理 bench_other 与 test_utils 死代码：第三 commit 进一步删除 bench_other 变体，并清理 sglang.test.test_utils 中仅为这些脚本服务的 helper，保持工具函数与调用方一致。
4. 引用验证：作者声明 Verified no dangling references remain after removal，确保删除后无任何残留引用。
5. CI 与测试配套：本 PR 无新增测试，作者通过 /tag-and-rerun-ci 触发 CI；PR states 显示 Base 无结果、Extra 有一次失败记录，但未阻塞合入。

关键文件：

- benchmark/json_schema/bench_sglang.py（模块 JSON 基准；类别 source；类型 deletion；符号 schema_gen, contains_formats, convert_dataset, bench_schema）：该

文件是 JSON Schema 结构化输出基准的入口，删除整条评测链路（schema_gen/bench_schema），是本次清理中最有代表性的 SGLang DSL 基准之一。

- benchmark/generative_agents/agent_functions.py（模块 智能体代理；类别 source；类型 deletion；符号 poignancy_event, poignancy_event_prompt, generate_event_triple, generate_event_triple_prompt）：复刻 generative_agents 论文的 top 5 agent 函数，同时提供 @sgl.function 与纯 prompt 两套实现，展示了早期 DSL 与 prompt 工程对比。
- benchmark/json_jump_forward/bench_other.py（模块 JSON 跳转；类别 source；类型 deletion；符号 character_gen, city_gen, character_maker, call_generate_lmql）：依赖 guidance/lmql 的第三方对比脚本，因第三方依赖不可运行而被删除。
- benchmark/reasoning_benchmark/answer_extraction.py（模块 推理基准；类别 source；类型 deletion；符号 _fix_fracs, _fix_a_slash_b, _fix_sqrt, _fix_tan）：从 DeepSeek-Math 移植的数学答案抽取器，包含 LaTeX 修复逻辑，证明该 benchmark 属于历史评估工具。
- benchmark/benchmark_batch/benchmark_tokenizer.py（模块 批量基准；类别 source；类型 deletion；符号 main, run_benchmark, benchmark, print_results）：曾用于测量 patch_tokenizer 顺序 vs 批处理性能，属于一次性验证脚本。
- benchmark/tree_of_thought_deep/bench_other.py（模块 思维树；类别 source；类型 deletion；符号 get_answer_value, most_frequent_number, propose_plan, execute_plan）：多分支思维树（ToT）基准实现，包含 tree_search 编排，是删除的典型 agentic benchmark。
- benchmark/reasoning_benchmark/eval_utils.py（模块 数学评测；类别 source；类型 deletion；符号 parse_digits, is_digit, symbolic_equal, _parse）：基于 sympy 的数学等价性判定，与 answer_extraction 配套。
- benchmark/react/bench_other.py（模块 代理脚本；类别 source；类型 deletion；符号 get_prompt, main, run_single_agent, run_single_agent_async）：ReAct 代理基准，依赖长 prompt 模板与 HotPotQA 数据集。
- benchmark/benchmark_batch/benchmark_batch.py（模块 批量请求；类别 source；类型 deletion；符号 generate_random_prompt, generate_random_text, prepare_all_prompts, send_batch_request）：批量请求基准脚本，硬编码 ENDPOINT_URL 与 TOKENIZER_DIR，属于一次性脚本。
- benchmark/dspy/bench_dspy_intro.py（模块 DSPy 集成；类别 source；类型 deletion；符号 BasicQA, GenerateAnswer, RAG, init）：来自 DSPy intro notebook 的改编，验证 sglang 作为 DSPy 后端，已无维护。

关键符号：schema_gen, bench_schema, contains_formats, convert_dataset, poignancy_event, action_location_sector, tree_search, propose_plan, execute_plan, reflect_solution, strip_string, extract_answer, math_equal, run_benchmark, benchmark, character_gen, city_gen, run_single_agent, send_batch_request, RAG.forward

关键源码片段

benchmark/json_schema/bench_sglang.py

该文件是 JSON Schema 结构化输出基准的入口，删除整条评测链路（schema_gen/bench_schema），是本次清理中最有代表性的 SGLang DSL 基准之一。

```
import json
import jsonschema
from datasets import load_dataset

import sglang as sgl

# 本文件整体删除；删除前负责 JSON Schema 结构化生成的评测。
# 核心是 schema_gen 声明式函数与 bench_schema 批量执行与校验。


@sgl.function
def schema_gen(s, message, json_schema):
    # 由 system + user 消息拼接 prompt，要求输出符合 json_schema
    system, user = message
    s += sgl.system(system)
    s += sgl.user(user)
    s += sgl.assistant(
        sgl.gen('json_output', temperature=0, max_tokens=256, json_schema=json_schema)
    )


def contains_formats(schema, formats):
    # 递归检查 schema 中是否包含指定 format（如 email）
    if isinstance(schema, dict):
        if schema.get('format', None) in formats:
            return True
        for value in schema.values():
            if contains_formats(value, formats):
                return True
    elif isinstance(schema, list):
        for item in schema:
            if contains_formats(item, formats):
                return True
    return False


def bench_schema(args):
    # 加载数据集并批量运行 schema_gen，随后逐条做 jsonschema 校验
    raw_dataset = load_dataset(args.data_path)
    arguments = []
    for data in raw_dataset['train']:
        messages = data['prompt']
        schema = data['schema']
        obj = json.loads(schema)
        # 跳过损坏样本与 outlines 不支持的 email 格式
        if obj.get('type') is None or contains_formats(obj, ['email']):
```

```
            continue
        system, user = messages
        arguments.append({
            'message': (json.dumps(system['content']), json.dumps(user['content'])),
            'json_schema': schema,
        })

    backend = select_sglang_backend(args)
    sgl.set_default_backend(backend)
    states = schema_gen.run_batch(
        arguments, temperature=0, num_threads=args.parallel
    )

    # 校验输出是否合法，记录失败项索引
    invalid = []
    for i, state in enumerate(states):
        try:
            schema = json.loads(arguments[i]['json_schema'])
            obj = json.loads(state['json_output'])
            assert jsonschema.validate(obj, schema) is None
        except Exception:
            invalid.append(i)

    return states, invalid
```

benchmark/benchmark_batch/benchmark_tokenizer.py

曾用于测量 patch_tokenizer 顺序 vs 批处理性能，属于一次性验证脚本。

```
from statistics import mean

# 本文件整体删除；用于对比 tokenizer 顺序处理与批处理耗时。
# 下方 benchmark 是单批次测量核心：分别记录顺序与批处理的多次运行时间。


def benchmark(*, data, batch_size, sequential_fn, batch_fn, num_runs, batch_mode):
    batch_data = data[:batch_size]
    run_single = 'single' in batch_mode
    run_batch = 'batch' in batch_mode

    out = {'batch_size': batch_size}

    if run_single:
        times = measure_times(fn=lambda: sequential_fn(batch_data), num_runs=num_runs)
        out['avg_sequential_ms'] = mean(times)
        out['sequential_runs'] = times

    if run_batch:
        times = measure_times(fn=lambda: batch_fn(batch_data), num_runs=num_runs)
        out['avg_batch_ms'] = mean(times)
```

```
        out['batch_runs'] = times

    # 两种模式都跑时计算加速比
    if run_single and run_batch and out['avg_batch_ms'] > 0:
        out['speedup_factor'] = out['avg_sequential_ms'] / out['avg_batch_ms']

    return out
```

benchmark/tree_of_thought_deep/bench_other.py

多分支思维树（ToT）基准实现，包含 tree_search 编排，是删除的典型 agentic benchmark。

```
# 本文件整体删除；实现多分支思维树搜索：规划、执行、反思、定稿。
# 删除原因：依赖 guidance/lmql 分支已不可运行，且仓库内无引用。

USER_PREFIX = '[INST] '
USER_SUFFIX = ' [/INST]'
ASSISTANT_SUFFIX = ' </s><s>'
temp = 0.001


def propose_plan(s, question, num_branches, call_generate):
    # 首先生成 num_branches 个高层面解题计划
    plan_prompt = (
        'Please generate a high-level plan for solving the following question. '
        'Keep your response concise and within 80 words. Question: '
    )
    s += USER_PREFIX + plan_prompt + question + USER_SUFFIX
    comps = call_generate(
        s, max_tokens=256, temperature=temp, stop=None, n=num_branches
    )
    return [s + comp + ASSISTANT_SUFFIX for comp in comps]


def tree_search(question, num_branches, call_generate):
    # 思维树主流程：plan -> execute -> reflect -> final answer，逐层分叉
    plan_forks = propose_plan('', question, num_branches, call_generate)

    sol_states = []
    for plan in plan_forks:
        # execute_plan 按规划逐步计算，并返回新的分支状态
        sol_states.extend(execute_plan(plan, num_branches, call_generate))

    ref_states = []
    for sol in sol_states:
        # reflect_solution 让模型自评打分，随后 get_final_answer 收敛最终答案
        ref_states.extend(reflect_solution(sol, num_branches, call_generate))

    solutions = []
    for sol in ref_states:
```

```
            solutions.append(get_final_answer(sol, num_branches, call_generate))

    return solutions
```

评论区精华

review 线程为空，唯一评论是作者自己的 /tag-and-rerun-ci 指令，用于重新触发 CI。没有 reviewer 对删除范围、判定标准或 test_utils helper 清理提出异议。整个清理决策由作者独立完成并自行合入，缺少外部视角对"长期无人维护"判定的复核。

- CI 重跑触发 (other): 仅完成 CI 重跑，未产生技术讨论；删除决策由作者自行确认。

风险与影响

- 风险：低风险，但有三点需要留意：
  - 外部引用无法完全排除：PR 验证的是仓库内部引用，仓库外用户脚本若 import 这些 benchmark 模块，升级后会直接失效。
  - test_utils helper 清理：若存在通过字符串拼接或间接导入引用已删 helper 的测试，静态搜索可能漏报；作者通过 CI 验证了现有测试。
  - 历史资产需从 Git 找回：answer_extraction.py、eval_utils.py 这类工具若未来重建 reasoning benchmark，需要重新移植。
  - 影响：仓库净删除约 8429 行、涉及 81 个文件；22 个 benchmark 目录从 main 分支消失。对 sglang 运行时、调度、kernel 等核心路径零影响。对依赖这些脚本的开发者是破坏性变化，但可通过 git 历史找回。对团队而言降低了 benchmark 资产维护面，使剩余 benchmark 更聚焦到 CI 与文档实际引用的路径。
  - 风险标记：纯删除变更，外部引用风险，test_utils 同步清理，历史脚本需从 Git 找回

关联脉络

- PR #34423 [diffusion] fix: nightly diffusion benchmark passes the retired --warmup flag: 同属 benchmark/CI 维护线，清理弃用参数与未维护基准
- PR #34464 Refocus LoRA tests on regression coverage: 同一时期测试 / 基准资产精简方向，聚焦回归契约