PR #27433 完整报告

sgl-project/sglang

[NPU] Nightly CI refactor and enhancement

合并时间：2026-06-26 08:52

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27433

执行摘要

- 一句话：NPU 夜间 CI 指标仪表板和可重用工作流
- 推荐动作：该 PR 适合 NPU 相关开发者和 CI 运维人员精读。值得关注的设计决策包括：
  - 基于 stdout 模式匹配的轻量级指标收集方案，无需额外服务。
  - 可重用工作流模式（单节点、多节点）降低重复配置。
  - 优雅处理测试失败（即使失败也生成指标工件）确保仪表板完整性。
  - 自动网络接口选择逻辑，对多机部署有参考价值。

功能与动机

现有的 NPU 夜间测试工作流只输出简单的通过 / 失败门控，没有每个测试结果的可视性。当失败发生时，必须手动检查各个作业日志以确定哪些模型或配置回归了。此外，测试指标值及其基线不以结构化方式捕获或比较，使得难以跟踪性能趋势。

实现拆解

1. 重构 .github/workflows/nightly-test-npu.yml，将其拆分为可重用工作流编排：单节点测试、多节点 PD 分离和多节点混合部署。添加 check-all-jobs 门控，通过 GITHUB_STEP_SUMMARY 输出指标仪表板，包含每个测试的状态、测量值和基线对比。
2. 新增 .github/workflows/nightly-test-npu-e2e-single-node.yml 可重用工作流，使用 || test_exit_code=$? 优雅处理测试失败，确保即使失败也生成 metrics.json 并上传为工件（if: always()）。
3. 新增 .github/workflows/nightly-test-npu-e2e-multi-node.yml 可重用工作流，通过 Kubernetes 编排多节点测试，支持 PD 分离和混合部署模式。
4. 添加 Python 测试工具库：
   - test_npu_performance_utils.py：基类定义 assert_metrics()，通过 dump_metric() 输出吞吐量、TPOT、TTFT、E2E 延迟及其基线。
   - test_npu_accuracy_utils.py：基类定义 run_evalscope() 和 assert_metrics()，转储准确率和基线。
   - gen_dataset_fixed_len.py：生成固定 token 长度数据集的工具，支持 GSM8K 和多模态数据。

- run_npu_e2e_test.py：K8s 资源管理，包括创建 Pod、Job、ConfigMap、Role 等，支持 Jinja2 模板。
- test_npu_multi_node_utils.py：多节点辅助函数，如自动检测网络接口、角色启动、环境变量设置。

5. 新增多个具体测试用例（约 15 个文件），覆盖 Qwen3、DeepSeek、MiniMax、GLM5 等模型系列，验证不同精度、输入长度、前缀缓存和多节点配置下的性能和准确率。所有测试用例通过 register_npu_ci 注册到 CI 框架。

关键文件：

- .github/workflows/nightly-test-npu.yml（模块 CI 流程；类别 infra；类型 ci-config）：主 CI 工作流，重构为门控仪表板，编排所有测试组，是本次变更的核心调度文件。
- python/sglang/test/ascend/e2e/test_npu_performance_utils.py（模块 性能测试工具；类别 test；类型 test-coverage；符号 retry, get_cann_version, write_pkg_info_to_file, run_bench_serving）：性能测试基类，定义 assert_metrics 和指标转储方法，是所有性能测试的公共依赖。
- python/sglang/test/ascend/e2e/gen_dataset_fixed_len.py（模块 数据集生成；类别 test；类型 test-coverage；符号 load_jsonl, save_jsonl, format_qa, pad_to_target_tokens）：数据集生成工具，用于创建固定 token 长度输入，支撑多种测试场景。
- python/sglang/test/ascend/e2e/test_npu_accuracy_utils.py（模块 准确率测试工具；类别 test；类型 test-coverage；符号 run_evalscope, assert_metrics, TestNpuAccuracyTestCaseBase, setUpClass）：准确率测试基类，封装 evalscope 调用和断言逻辑。
- python/sglang/test/ascend/e2e/run_npu_e2e_test.py（模块 测试编排；类别 test；类型 test-coverage；符号 get_unique_random_string, create_kube_yaml, create_pod, delete_pod）：K8s 编排脚本，支持创建 / 删除 Pod、Job、ConfigMap 等，是多节点测试的关键组件。
- .github/workflows/nightly-test-npu-e2e-single-node.yml（模块 单节点工作流；类别 infra；类型 ci-config）：单节点端到端可重用工作流，定义指标收集和工件上传流程。
- python/sglang/test/ascend/e2e/test_npu_multi_node_utils.py（模块 多节点工具；类别 test；类型 test-coverage；符号 get_nic_name, get_host_name, get_host_ip, get_k8s_api）：多节点测试辅助函数，包括网络接口自动选择、角色启动、ConfigMap 查询等。

关键符号：assert_metrics, run_evalscope, pad_to_target_tokens, create_pod, get_nic_name, run_bench_serving

关键源码片段

python/sglang/test/ascend/e2e/gen_dataset_fixed_len.py

数据集生成工具，用于创建固定 token 长度输入，支撑多种测试场景。

```
def pad_to_target_tokens(
    question,
    few_shot_pool_token_ids,
```

```
    tokenizer,
    target_tokens,
    test_template="Question: {question}\nLet's think step by step\nAnswer:\n",
):
    """Pad a question text to the target token length."""
    # 生成测试 prompt 并获取其 token ID
    test_prompt = test_template.format(question=question)
    test_token_ids = tokenizer.encode(test_prompt, add_special_tokens=False)

    # 计算还需要填充的 token 数
    remaining_tokens = target_tokens - len(test_token_ids)
    if remaining_tokens <= 0:
        # 如果已经超出目标长度，直接截断
        return tokenizer.decode(test_token_ids[:target_tokens], skip_special_tokens=True)

    # 随机打乱 few-shot pool 索引
    shuffled_ids = list(range(len(few_shot_pool_token_ids)))
    random.shuffle(shuffled_ids)

    # 从 pool 中抽取 token 填充到 prefix
    prefix_ids = []
    for idx in shuffled_ids:
        fs_ids = few_shot_pool_token_ids[idx]
        if len(prefix_ids) + len(fs_ids) <= remaining_tokens:
            prefix_ids.extend(fs_ids)
        else:
            # 如果超出剩余空间，只取部分
            partial_gap = remaining_tokens - len(prefix_ids)
            if partial_gap > 0:
                prefix_ids.extend(fs_ids[:partial_gap])
            break

    # 如果 pool 不够填充，重复第一个样本直到填满
    if len(prefix_ids) < remaining_tokens and few_shot_pool_token_ids:
        padding_source_ids = few_shot_pool_token_ids[shuffled_ids[0]]
        repeat_count = (remaining_tokens // len(padding_source_ids)) + 1
        padding_ids = (padding_source_ids * repeat_count)[:remaining_tokens - len(prefix_ids)]
        prefix_ids.extend(padding_ids)

    # 拼接 prefix 和测试 prompt，并解码
    full_ids = prefix_ids + test_token_ids
    return tokenizer.decode(full_ids[:target_tokens], skip_special_tokens=True)
```

评论区精华

Review 中主要讨论了以下问题：

- 工作流文件位置：iforgetmyname 建议将可重用工作流移到 python/sglang/test/ascend，但作者指出 GitHub Actions 要求本地工作流引用必须在 .github/workflows 下，因此维持

原位置。

- 多节点镜像架构：iforgetmyname 指出多节点模板默认使用 linux-aarch64-a3-0 runner 但容器镜像是 x86，作者解释 CPU 控制节点是 x86 架构，因此使用 x86 镜像，但 runner 标签应改为 linux-amd64-cpu-8，后续提交中已修正。
- 删除冗余文件：iforgetmyname 要求删除 run_aisbench.sh，作者确认删除。
- 环境变量清理：iforgetmyname 询问是否应移除工作流中的某个环境变量，作者后续提交已调整。
- 工作流文件应放在何处 (design): 作者指出 GitHub Actions 要求本地工作流引用必须位于 .github/workflows，因此维持原位。
- 多节点默认 runner 架构与容器镜像不匹配 (correctness): 作者说明 CPU 控制节点是 x86 架构，后续将 runner 改为 linux-amd64-cpu-8。
- 删除未使用的脚本文件 (other): 作者确认删除。
- 清除不必要的环境变量 (question): 作者在后续提交中已调整，推测已解决。

风险与影响

- 风险：
  - 配置风险：大量新增工作流和测试用例可能引入 YAML 或脚本错误，导致夜间测试不稳定。
  - K8s 依赖：多节点测试依赖外部 Kubernetes 集群，集群不可用或资源不足将导致测试失败。
  - 基线硬编码：基线值硬编码在测试文件中（如 tpot_baseline: 0.05），模型升级或环境变化后可能需手动更新，否则可能误报回归。
  - 网络接口检测：get_nic_name() 基于 /proc/net/dev 接口流量选择，在某些虚拟化环境中可能选错，导致多机通信失败。
  - 数据集生成：gen_dataset_fixed_len.py 的 token 填充逻辑假设 tokenizer 兼容，未测试所有模型 tokenizer。
  - 影响：影响范围：仅影响 NPU 夜间 CI 流程，不涉及 SRT 核心推理代码。
  - 对用户：无直接影响。
  - 对系统：夜间测试从简单 pass/fail 升级为结构化指标仪表板，显著提升可观察性。
  - 对团队：NPU 团队能更快定位性能回归和准确率退化；CI 运维人员需维护 K8s 集群和环境。
  - 影响程度：中等，因为涉及大量新配置和外部依赖。
  - 风险标记：K8s 外部依赖，基线硬编码，多节点配置复杂，大量新增配置可能不稳定

关联脉络

- 暂无明显关联 PR