# PR #22965 完整报告

sgl-project/sglang

migrate CPU-only unit tests from openai_server to unit/

合并时间：2026-04-16 18:53

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/22965

## 执行摘要

- 一句话：将 OpenAI 端点 CPU-only 单元测试迁移至专用目录并注册到 CPU CI 阶段。
- 推荐动作：该 PR 对于负责测试基础设施和 CI 优化的工程师值得精读，关注点包括：测试组织策略（如何分离 CPU/GPU 测试）、mock 使用技巧（如 test_dpsk_v32_encoding_path 中的简化），以及 CI 注册配置的调整。设计决策展示了如何通过代码迁移和 stub 提高测试兼容性。

## 功能与动机

根据 PR body 描述，目的是“Move CPU-only test classes out of GPU test suites into test/registered/unit/entrypoints/openai/, registered under stage-a-test-cpu。”这旨在分离测试逻辑，将不依赖 GPU 的单元测试从 GPU 测试套件中移出，以避免不必要的资源消耗和简化测试分类，从而优化持续集成（CI）流程。

## 实现拆解

1. 创建新的单元测试文件：在 test/registered/unit/entrypoints/openai/test_matched_stop.py 中新增 TestRegexPatternMaxLength 测试类，专门验证正则表达式最大长度计算。使用 register_cpu_ci 注册到 stage-a-test-cpu 套件，原因是将 CPU-only 逻辑从 GPU 依赖中分离，确保测试在纯 CPU 环境下运行。
2. 修改原有测试文件：从 test/registered/openai_server/validation/test_matched_stop.py 移除 TestRegexPatternMaxLength 类，保持该文件专注于 GPU 测试，简化职责并避免重复。
3. 迁移其他测试文件：将 test_serving_chat.py、test_serving_completions.py、test_protocol.py 从 openai_server/basic/ 移动到 unit/entrypoints/openai/，并更新导入语句。将 CI 注册从 register_cuda_ci 和 register_amd_ci 替换为 register_cpu_ci，因为这些测试不依赖 GPU。
4. 添加兼容性代码：在每个迁移的文件开头添加 from sglang.test.test_utils import maybe_stub_sgl_kernel 和 maybe_stub_sgl_kernel() 调用，以防止导入 sgl_kernel 时触发 GPU 依赖，确保测试在 CPU 环境下兼容。
5. 优化 mock 逻辑：在 test_serving_chat.py 的 test_dpsk_v32_encoding_path 测试中，用 _MockTokenizerManager mock 对象替换真实的 ServerArgs 初始化，简化测试并避免环境依赖，提高可维护性。

关键文件：

- test/registered/unit/entrypoints/openai/test_matched_stop.py（模块 匹配停止测试；类别 test；类型 test-coverage；符号 TestRegexPatternMaxLength, setUpClass, test_get_max_length）：新增文件，包含专门的 TestRegexPatternMaxLength 测试类，用于验证正则表达式最大长度计算，是 CPU-only 测试迁移的核心部分。
- test/registered/openai_server/validation/test_matched_stop.py（模块 匹配停止测试；类别 test；类型 test-coverage；符号 TestRegexPatternMaxLength, setUpClass, test_get_max_length）：修改文件，移除了 TestRegexPatternMaxLength 类，使该文件专注于 GPU 相关的匹配停止测试，简化职责。
- test/registered/unit/entrypoints/openai/test_serving_chat.py（模块 聊天服务测试；类别 test；类型 rename-or-move）：重命名并迁移文件，包含 OpenAI 聊天服务的单元测试，关键优化了 test_dpsk_v32_encoding_path 测试的 mock 逻辑。
- test/registered/unit/entrypoints/openai/test_serving_completions.py（模块 补全服务测试；类别 test；类型 rename-or-move）：重命名并迁移文件，包含 OpenAI 补全服务的单元测试，更新 CI 注册并添加兼容性代码。
- test/registered/unit/entrypoints/openai/test_protocol.py（模块 协议测试；类别 test；类型 rename-or-move）：重命名并迁移文件，包含 OpenAI 协议相关的单元测试，简化 CI 注册。

关键符号：TestRegexPatternMaxLength, setUpClass, test_get_max_length

## 关键源码片段

### test/registered/unit/entrypoints/openai/test_matched_stop.py

新增文件，包含专门的 TestRegexPatternMaxLength 测试类，用于验证正则表达式最大长度计算，是 CPU-only 测试迁移的核心部分。

```
import unittest

from sglang.srt.sampling.sampling_params import MAX_LEN, get_max_seq_length
from sglang.test.ci.ci_register import register_cpu_ci

# 注册到 CPU CI 阶段，指定估计时间和套件，确保测试在纯 CPU 环境下运行
register_cpu_ci(est_time=2, suite="stage-a-test-cpu")

class TestRegexPatternMaxLength(unittest.TestCase):
    """测试正则表达式模式的最大长度计算，从 GPU 套件中分离出来。"""
    @classmethod
    def setUpClass(cls):
        # 定义一组正则表达式字符串和预期最大长度的映射，涵盖无限重复、嵌套和分支等复杂情况
        cls.regex_str_to_max_len = {
            "((ablcd(elf){2}){3,5}glhij)*k": MAX_LEN, # 包含 '*' 重复，表示无限长度，需要特殊处理
            "abc*?k": MAX_LEN, # 懒惰匹配 '*' 仍需要无限存储，验证函数处理
            "^spec(foolat)$": 7, # '^' 和 '$' 不增加字符长度，计算基础字符串和选择
            "(a(bcalde(fglhi){2,3})j){2}kl": 22, # 复杂嵌套和重复，展示最大长度累加逻辑
            "(foo(barlbaz(qux){1,2}))l(x(yz){5,10})": 21, # 分支选择，取各分支最大值
```

```
        "(((a|bc){1,3}(d(e|f){2}|gh){2,4})|(ijk|lmp(no|p){3})){5}": 90, # 多层嵌套和重复，综合验证
    }

    def test_get_max_length(self):
        """遍历映射表，调用 get_max_seq_length 并断言结果与预期一致。"""
        for regex_str, max_len in self.regex_str_to_max_len.items():
            if max_len == MAX_LEN:
                # 对于无限长度，确保函数返回值至少为 MAX_LEN，避免错误截断
                self.assertGreaterEqual(get_max_seq_length(regex_str), MAX_LEN)
            else:
                # 对于有限长度，精确匹配预期值，验证计算准确性
                self.assertEqual(get_max_seq_length(regex_str), max_len)

if __name__ == "__main__":
    unittest.main()
```

## test/registered/unit/entrypoints/openai/test_serving_chat.py

重命名并迁移文件，包含 OpenAI 聊天服务的单元测试，关键优化了 test_dpsk_v32_encoding_path 测试的 mock 逻辑。

```
from sglang.test.test_utils import maybe_stub_sgl_kernel

# 在导入任何可能触发 GPU 依赖的模块前调用，确保测试在 CPU 环境下兼容
maybe_stub_sgl_kernel()

import json
import unittest
from sglang.test.ci.ci_register import register_cpu_ci

# 注册到 CPU CI 阶段，替换原有的 GPU 注册，明确测试不依赖 GPU
register_cpu_ci(est_time=8, suite="stage-a-test-cpu")

# 在 test_dpsk_v32_encoding_path 测试中，优化 mock 逻辑：
def test_dpsk_v32_encoding_path(self):
    """测试 DeepSeek V3.2 编码路径检测，使用简化的 mock 避免真实 ServerArgs 初始化。"""
    from sglang.srt.managers.template_manager import TemplateManager
    tm = _MockTokenizerManager() # 使用内部 mock 类，而不是真实 TokenizerManager
    mock_hf_config = Mock()
    mock_hf_config.architectures = ["DeepseekV32ForCausalLM"]
    tm.model_config.hf_config = mock_hf_config

    # 案例1：无聊天模板 + DeepSeek V3.2 架构 -> 应使用 dpsk 编码
    tm.tokenizer.chat_template = None
    serving_chat = OpenAIServingChat(tm, TemplateManager())
    self.assertTrue(serving_chat.use_dpsk_v32_encoding)

    # 案例2：有聊天模板 -> 不应使用 dpsk 编码
    tm.tokenizer.chat_template = "some template"
    serving_chat = OpenAIServingChat(tm, TemplateManager())
```

```
self.assertFalse(serving_chat.use_dpsk_v32_encoding)

# 案例3：非 DeepSeek V3.2 架构 -> 不应使用 dpsk 编码
mock_hf_config.architectures = ["LlamaForCausalLM"]
serving_chat = OpenAIServingChat(tm, TemplateManager())
self.assertFalse(serving_chat.use_dpsk_v32_encoding)
```

## 评论区精华

review 中提出了两个主要建议：

- 文件名误导：gemini-code-assist[bot] 指出 test_matched_stop.py 文件名与内容不符，因为文件现在只包含 TestRegexPatternMaxLength 测试，建议重命名为 test_regex_max_length.py 以更准确反映用途。
- docstring 过时：同一评论者发现 test_serving_chat.py 中的执行指令指向旧路径 tests/test_serving_chat_unit.py，建议更新以匹配新位置。 目前未看到这些建议被采纳或讨论结论，因此疑虑仍处于未解决状态。
- 文件名 test_matched_stop.py 可能误导 (style)：未在提供材料中看到采纳或进一步讨论，疑虑未解决。
- docstring 中的执行指令过时 (documentation)：未在提供材料中看到修改，疑虑未解决。

## 风险与影响

- 风险：- 回归风险：测试文件迁移和 mock 调整可能导致现有测试失败，例如 test_serving_chat.py 中的 test_dpsk_v32_encoding_path 测试简化后可能覆盖不全面。
- 兼容性风险：新增的 maybe_stub_sgl_kernel() 调用在不同环境（如本地开发或特定 CI 配置）中可能行为不一致，影响测试稳定性。
- CI 配置风险：将测试注册到 stage-a-test-cpu 可能改变 CI 流水线的执行顺序或资源分配，需确保与其他测试阶段协调。
- 影响：- 用户影响：无直接影响，这是内部测试基础设施变更。
- 系统影响：测试执行更高效，CPU-only 测试独立运行，减少 GPU 资源占用，可能加快 CI 反馈周期。
- 团队影响：工程师需要更新本地测试运行命令以适应新路径，但长期看简化了测试维护和分类。
- 风险标记：测试覆盖调整 , CI 配置变更 , mock 不完整风险

## 关联脉络

- PR #22910 ci: re-enable fp8 nightly benchmark configs: 同样涉及 CI 配置调整，虽然针对 GPU 基准测试，但展示了仓库中测试基础设施的持续优化趋势。
- PR #22926 [misc] Configure logging before ServerArgs.post_init: 同为重构类型 PR，涉及代码组织调整，可对比学习测试与配置的迁移策略。