PR #29791 完整报告

sgl-project/sglang

[diffusion] Add 5090 diffusion consumer GPU guard

合并时间：2026-07-01 19:09

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29791

执行摘要

- 一句话：为 RTX 5090 添加扩散模型 CI 防护
- 推荐动作：值得精读的设计决策：平台感知的基线拆分模式（别名字典、环境变量覆盖、自动检测 fallback）为未来添加更多 GPU 变体提供了清晰模式。GT 查找的 'platform → default' 递进策略也是可复用的。团队在添加新硬件时可直接参考此 PR 的结构。

功能与动机

PR 描述指出需要为 1-gpu-5090 消费级 GPU 添加 CI 防护，避免类似 #18997 的低 VRAM 失败模式，并防止 TurboWan 稀疏注意力后端在消费级 GPU 上回退行为异常。

实现拆解

1. 构建平台感知测试工具：在 test_utils.py 中添加 CONSISTENCY_PLATFORM_ENV、CONSISTENCY_THRESHOLD_DIR、CONSISTENCY_THRESHOLD_FILE_BY_PLATFORM 和 CONSISTENCY_PLATFORM_ALIASES，新增 get_consistency_platform() 根据硬件自动选择平台或通过环境变量覆盖，新增 get_consistency_threshold_path() 返回对应平台阈值文件路径，新增 get_consistency_gt_candidates() 支持平台子目录优先查找。
2. 拆分性能基线文件：在 test/server/perf_baselines/ 下按平台拆分原 perf_baselines.json 为 h100.json、b200.json、5090.json，在 testcase_configs.py 中添加 PERF_BASELINE_FILE_BY_PLATFORM、PERF_BASELINE_PLATFORM_ALIASES 和 get_perf_baseline_path() 动态加载对应平台基线，BaselineConfig.load() 现调用 get_perf_baseline_path()。
3. 拆分一致性阈值文件：在 test/server/consistency_thresholds/ 下按平台存放阈值文件（h100.json、b200.json、5090.json），并添加 _merge_threshold_metadata() 实现平台覆盖与基础配置的合并。
4. 定义 5090 测试用例集：在 gpu_cases.py 中添加 _select_5090_canary_cases() 从 ONE_GPU_CASES 中选取 4 个代表用例（zimage、flux2、wan2.1、turbo wan2.1），并根据白名单启用性能检查、跳过已知不稳定的 consistency 用例；新增 _make_5090_flux_layerwise_cpu_offload_case() 创建针对低 VRAM 的 offload 测试用例（启用 --dit-cpu-offload、--dit-layerwise-offload、--pin-cpu-memory），输出尺寸缩小为 512x512 以减少显存。

5. 添加 TurboWan 后端选择单元测试：新增 test_turbo_wan_backend.py，覆盖 _resolve_turbo_wan_sparse_backend() 的 5 种场景：非稀疏请求回退、sage_sla 类型优先、显式请求被尊重、supported_backends 过滤、空交集保留原选择。
6. 更新 CI 工作流：在 pr-test-multimodal-gen.yml 中添加 1-gpu-5090 作业，指定使用 5090-d-runner 标签，运行 test_server_1_gpu_5090.py 和 test_turbo_wan_backend.py；同时更新 diffusion_case_parser.py 以支持多基线文件遍历。

关键文件：

- python/sglang/multimodal_gen/test/test_utils.py（模块 测试工具；类别 test；类型 test-coverage；符号 _load_threshold_json, _normalize_consistency_platform, get_consistency_platform, get_consistency_threshold_path）：核心测试工具，实现平台感知的一致性阈值加载、GT 候选生成，是整个多平台基线机制的基础。
- python/sglang/multimodal_gen/test/server/gpu_cases.py（模块 测试用例；类别 test；类型 test-coverage；符号 _select_5090_canary_cases, _make_5090_flux_layerwise_cpu_offload_case）：定义了 5090 测试用例集合，包括选择 and 定制化 offload 用例，是 5090 CI 作业的核心驱动。
- python/sglang/multimodal_gen/test/unit/test_turbo_wan_backend.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestTurboWanBackendSelection, test_non_sparse_requested_backend_falls_back_to_attention_type, test_sagesla_attention_type_prefers_sage_sparse_backend, test_requested_sparse_backend_is_honored）：新增的 TurboWan 后端选择单元测试，覆盖了 5 种关键场景，确保稀疏注意力后端回退行为正确。
- python/sglang/multimodal_gen/test/server/testcase_configs.py（模块 配置加载；类别 test；类型 test-coverage；符号 _normalize_perf_baseline_platform, get_perf_baseline_platform, get_perf_baseline_path）：实现性能基线平台感知加载，新增 PERF_BASELINE_FILE_BY_PLATFORM 和 get_perf_baseline_path()，替换原有单文件加载。
- python/sglang/multimodal_gen/test/server/perf_baselines/5090.json（模块 性能基线；类别 test；类型 test-coverage）：新增 RTX 5090 性能基线数据，包含三个场景的 stage 耗时和 denoise step 数据，用于性能回归检测。
- python/sglang/multimodal_gen/test/server/consistency_thresholds/5090.json（模块 一致性阈值；类别 test；类型 test-coverage）：新增 5090 一致性阈值覆盖，为 zimage 用例设置更宽容的 PSNR 阈值（25 对比默认 28）。
- .github/workflows/pr-test-multimodal-gen.yml（模块 CI 配置；类别 infra；类型 infrastructure）：新增 1-gpu-5090 CI 作业，确保 5090 测试在 PR 中自动运行。

关键符号：get_consistency_platform, get_consistency_threshold_path, get_consistency_gt_candidates, _merge_threshold_metadata, _select_5090_canary_cases, _make_5090_flux_layerwise_cpu_offload_case, _resolve_turbo_wan_sparse_backend (tested), get_perf_baseline_platform, get_perf_baseline_path

关键源码片段

python/sglang/multimodal_gen/test/test_utils.py

核心测试工具，实现平台感知的一致性阈值加载、G候选生成，是整个多平台基线机制的基础。

```
import os
from pathlib import Path
from typing import Any

CONSISTENCY_PLATFORM_ENV = "SGLANG_DIFFUSION_CONSISTENCY_PLATFORM"
CONSISTENCY_THRESHOLD_DIR = (
    Path(__file__).resolve().parent / "server" / "consistency_thresholds"
)
CONSISTENCY_THRESHOLD_FILE_BY_PLATFORM = {
    "h100": "h100.json",
    "b200": "b200.json",
    "5090": "5090.json",
}
CONSISTENCY_PLATFORM_ALIASES = {
    "sm90": "h100", # 架构名 → 标准化平台名
    "hopper": "h100",
    "h100": "h100",
    "sm100": "b200",
    "blackwell": "b200",
    "b200": "b200",
    "sm120": "5090",
    "rtx5090": "5090",
    "5090": "5090",
}


def _normalize_consistency_platform(platform: str) -> str:
    """将用户输入的任意平台字符串标准化为内部键（如 '5090'）。"""
    normalized = platform.strip().lower().replace("_", "-").replace("-", "")
    if normalized not in CONSISTENCY_PLATFORM_ALIASES:
        raise ValueError(
            f"Invalid diffusion consistency platform {platform!r}. "
            f"Expected one of: {', '.join(sorted(CONSISTENCY_THRESHOLD_FILE_BY_PLATFORM))}
            "
        )
    return CONSISTENCY_PLATFORM_ALIASES[normalized]


def get_consistency_platform() -> str:
    """根据环境变量或当前硬件返回平台标识。"""
    override = os.getenv(CONSISTENCY_PLATFORM_ENV)
    if override:
        return _normalize_consistency_platform(override)
    if current_platform.is_sm120(): # sm120 对应 RTX 5090
        return "5090"
    if current_platform.is_blackwell():
```

```
        return "b200"
    return "h100" # 默认回退 H100


def get_consistency_threshold_path(platform: str | None = None) -> Path:
    """返回当前平台的一致性阈值 JSON 路径。"""
    threshold_platform = (
        _normalize_consistency_platform(platform)
        if platform is not None
        else get_consistency_platform()
    )
    return CONSISTENCY_THRESHOLD_DIR / CONSISTENCY_THRESHOLD_FILE_BY_
    PLATFORM[threshold_platform]


def _merge_threshold_metadata(base: dict[str, Any], override: dict[str, Any]) -> dict[str, Any]:
    """合并基础阈值配置与平台覆盖配置，cases 字段按 key 逐项覆盖。"""
    merged = dict(base)
    if "cases" in base or "cases" in override:
        merged["cases"] = {
            **base.get("cases", {}),
            **override.get("cases", {}),
        }
    for key, value in override.items():
        if key != "cases":
            merged[key] = value
    return merged
```

python/sglang/multimodal_gen/test/server/gpu_cases.py

定义了 5090 测试用例集合，包括选择 and 定制化 offload 用例，是 5090 CI 作业的核心驱动。

```
ONE_GPU_5090_PERF_CASE_IDS = frozenset({
    "zimage_image_t2i",
    "flux_2_klein_base_image_t2i",
    "wan2_1_t2v_1.3b",
})
ONE_GPU_5090_SKIP_CONSISTENCY_CASE_IDS = frozenset({
    "turbo_wan2_1_t2v_1.3b", # 5090 上产生视觉不同帧，跳过 GT 比较
})


def _select_5090_canary_cases(case_ids: tuple[str, ...]) -> list[DiffusionTestCase]:
    """从 ONE_GPU_CASES 中选取指定用例，并覆写 run_perf_check 和 run_consistency_
    check。"""
    cases_by_id = {case.id: case for case in ONE_GPU_CASES}
    missing = [case_id for case_id in case_ids if case_id not in cases_by_id]
    if missing:
        raise RuntimeError(f"Unknown 5090 diffusion canary case(s): {missing}")
```

```
    return [
        replace(
            cases_by_id[case_id],
            # 仅白名单用例启用性能检查
            run_perf_check=case_id in ONE_GPU_5090_PERF_CASE_IDS,
            # 保留原一致性设置，但在黑名单中则强制关闭
            run_consistency_check=(
                cases_by_id[case_id].run_consistency_check
                and case_id not in ONE_GPU_5090_SKIP_CONSISTENCY_CASE_IDS
            ),
        )
        for case_id in case_ids
    ]


def _make_5090_flux_layerwise_cpu_offload_case() -> DiffusionTestCase:
    """创建针对 5090 低 VRAM 的 Flux 层 offload 测试用例。"""
    base_case = next(case for case in ONE_GPU_CASES if case.id == "flux_image_t2i")
    return replace(
        base_case,
        id="flux_image_t2i_layerwise_cpu_offload_5090",
        server_args=replace(
            base_case.server_args,
            dit_layerwise_offload=True,
            dit_offload_prefetch_size=5,
            text_encoder_cpu_offload=True,
            extras=[
                *base_case.server_args.extras,
                "--dit-cpu-offload",
                "--pin-cpu-memory",
            ],
        ),
        # 缩小输出尺寸以适配 24 GB VRAM
        sampling_params=replace(
            T2I_sampling_params,
            output_size="512x512",
            extras={"num_inference_steps": 4, "seed": 0},
        ),
        # 仅验证不崩溃，不检查性能和一致性
        run_perf_check=False,
        run_consistency_check=False,
        run_component_accuracy_check=False,
        run_models_api_check=False,
        run_t2v_input_reference_check=False,
    )


ONE_GPU_5090_CASES = _select_5090_canary_cases(
    ("zimage_image_t2i", "flux_2_klein_base_image_t2i",
```

```
        "wan2_1_t2v_1.3b", "turbo_wan2_1_t2v_1.3b")
    )
    ONE_GPU_5090_CASES.append(_make_5090_flux_layerwise_cpu_offload_case())

    # 注册到参数化测试组
    PARAMETRIZED_CASE_GROUPS["1-gpu-5090"] = [
        ("test_server_1_gpu_5090.py", ONE_GPU_5090_CASES),
    ]
```

评论区精华

该 PR 无实质性 review 讨论，仅包含 bot 自动评论和触发 CI 的维护操作。PR 描述明确指出了 5090 的 consistency 策略：仅对 zimage 保留一致性检查（略低 PSNR 阈值），对 wan2.1 跳过一致性比较直到生成专用 GT。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 平台检测健壮性：依赖 current_platform.is_sm120() 和 is_blackwell() 等函数，若新硬件未及时更新会导致加载错误基线或阈值。
  2. 基线数据可靠性：5090 性能基线来自 CI 单次运行（run 28492141274），样本单一，后续硬件或驱动变更可能使基线失效。
  3. CI 耗时增加：最长用例估计约 329s，会增加 PR 测试流水线总时间，可能影响开发迭代速度。
  4. 配置维护成本：多平台基线和阈值文件需要团队持续更新和同步，增加运维负担。 - 影响：直接影响 diffusion 测试套件，使 5090 消费级 GPU 获得 CI 覆盖，防止类似 #18997 的回归。对用户无直接影响，但提升了低 VRAM 场景下的可靠性。团队成员需维护多平台基线和阈值文件，增加运维成本。 - 风险标记：平台检测依赖硬件函数，基线数据来自单次运行，新增 CI 作业增加耗时，需要维护多平台配置文件

关联脉络

- PR #29805 Platform baseline split (PR 描述中提及): 该 PR 折叠了 #29805 的平台基线拆分方案，二者共同构成了多平台 diffuusion 测试基础设施。