PR #35511 完整报告

sgl-project/sglang

[diffusion] CI: add minimax-h3 ref2va audio consistency coverage and guard peak vram

合并时间：2026-08-20 17:22

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35511

执行摘要

- 一句话：MiniMax-H3 Ref2VA 音频一致性 CI 覆盖与显存峰值守卫
- 推荐动作：值得精读。重点看两处设计：一是 load/runtime 显存峰值分离的思路与 replica all-reduce(MAX) 聚合的契约（gpu_worker.py + test_performance_metrics.py 对照阅读），这是 diffusion CI 显存门禁的通用范式；二是音频一致性度量组合（谱相似度 + 波形相关 +RMS+ 时长），可直接复用于其他 T2A 模型。若团队在维护 diffusion 模型，建议关注 h100.json 基线标定方式与缺基线时自动 dump 的降级策略。

功能与动机

PR 描述明确指出："This covers the Ref2VA reference-audio path missed by the existing mocked unit test and FL2VA/T2VA server cases." 现有单测只 mock 音频编码路径，服务端用例仅覆盖 FL2VA/T2VA，Ref2VA 的参考音频编码（依赖 ForwardContext 的 USPAttention 路径，源于 #34949 引入的回退）没有任何真实回归防护。同时作者观察到合并统计显存时 request-local 峰值 35,994 MiB 与包含加载的 replica 峰值 64,310 MiB 相差近 28 GiB，"demonstrating why load/runtime must be separated"，因此引入双峰值守卫，避免加载期 checkpoint 物化尖峰与 serving-time workspace/activation 回归互相掩盖。

实现拆解

1. 生产侧：GPUWorker 双显存峰值记录（gpu_worker.py）。在 __init__ 中于 init_device_and_model 返回后立即冻结 _load_peak_reserved_mb（CPU 平台置 0），并新增 _runtime_peak_reserved_mb 从零累计；_execute_forward_common 中 peak-reset 从 " 仅 output rank" 改为全体 rank 执行，_record_output_peak_memory 移到 _materialize_output_transport 之后并对所有 rank 更新 runtime 峰值；新增 _record_replica_peak_memory，在 collect_perf 且非 warmup 时用 get_replica_group().all_reduce(MAX) 聚合各 rank 的 load/runtime 峰值，写入每个请求的 memory_snapshots（load_peak / runtime_peak 两个快照键）。
2. 基线侧：per-scenario 显存基线接入（testcase_configs.py、test_server_utils.py、test_server_common.py、h100.json）。ScenarioConfig 新增 load_peak_vram_mb / runtime_peak_vram_mb 可选字段，h100.json 为所有 CUDA perf 用例补齐基线；PerformanceValidator.validate_peak_vram 使用独立容差（load 1% + 128 MiB、runtime 2% + 128 MiB）校验，缺基线或超限时先 _dump_baseline_for_testcase 落盘再 fail，保证一次 CI 运行就能产出新基线所需的全部数据，并把这组指标写入 pytest 报告、

合并 retry artifact 与 diffusion dashboard。

3. 音频一致性工具链（test_utils.py）。新增 AudioStreamInfo / probe_audio_stream（ffprobe 探测采样率、声道、时长）、extract_audio_pcm（ffmpeg 转 16 kHz 单声道 float32 PCM）、compare_audio_with_gt（谱相似度≥0.95、波形相关≥0.90、RMS 差≤2 dB、时长差≤0.10 s 四指标）、WAV GT 编解码与 artifact 保存，并支持平台目录优先的 GT 候选查找；同时把 SGL_TEST_FILES_CI_DATA_REVISION 升级到携带 MiniMax-H3 音频 GT 的提交。
4. Ref2VA E2E 服务端用例与回归单测。配置化新增 2-GPU MiniMax-H3 Ref2VA 用例，校验输出视频可解码且非静音；test_minimax_h3_media.py 新增 test_reference_audio_encode_sets_forward_context（FakeAudioVAE 复现 #35481 的 ForwardContext 回归）；test_performance_metrics.py 覆盖峰值分离、replica 聚合与独立容差，test_consistency_metrics.py 覆盖音频指标正反例与 GT 候选逻辑。
5. 配套基础设施。补充服务器启动失败清理（test_health_warmup_gate.py）、模型 revision 下载修复（test_hf_diffusers_utils.py、test_disagg_roles.py）、realtime 显存基线、Wan LoRA 基线校准等，保证新门禁不会因周边故障误报。

关键文件：

- python/sglang/multimodal_gen/runtime/managers/gpu_worker.py（模块 显存监控；类别 source；类型 core-logic；符号 _record_replica_peak_memory, _record_output_peak_memory, init）：唯一的生产代码变更：GPUWorker 新增 load/runtime 双显存峰值记录，并通过 replica group all-reduce(MAX) 聚合，是显存门禁的数据源头。
- python/sglang/multimodal_gen/test/test_utils.py（模块 测试工具；类别 test；类型 test-coverage；符号 AudioStreamInfo, probe_audio_stream, extract_audio_pcm, extract_audio_pcm_from_video_bytes）：新增整套音频一致性工具链（探测、PCM 抽取、四指标对比、WAV GT 编解码与 artifact 保存），并把 ci-data revision 升级到携带音频 GT 的提交。
- python/sglang/multimodal_gen/test/server/test_server_common.py（模块 服务端用例；类别 test；类型 test-coverage；符号 _validate_and_record, _validate_audio_consistency, _record_performance_result, _validate_realtime_performance）：把所有 CUDA perf 用例接入显存峰值门禁，并新增音频一致性校验分支；缺基线时自动 dump 并 fail，是 CI 门禁的落地关键。
- python/sglang/multimodal_gen/test/unit/test_performance_metrics.py（模块 性能指标；类别 test；类型 test-coverage；符号 _perf_record, test_performance_summary_separates_load_and_runtime_peaks, test_worker_records_replica_load_and_runtime_peaks, test_peak_vram_validation_uses_independent_tolerances）：新增文件，完整定义显存峰值分离、replica 聚合与独立容差校验的契约，是源码改动最直接的回归保障。
- python/sglang/multimodal_gen/test/unit/test_consistency_metrics.py（模块 一致性指标；类别 test；类型 test-coverage；符号 _sine_wave, _audio_thresholds, test_audio_consistency_wav_round_trip_passes, test_audio_consistency_rejects_wrong_audio）：为音频一致性工具链补充正反例单测（

WAV round-trip、错误音频拒绝、静音拒绝、平台目录优先等），验证阈值设计有效。

- python/sglang/multimodal_gen/test/unit/test_minimax_h3_media.py（模块 H3 媒体；类别 test；类型 test-coverage；符号 test_reference_audio_encode_sets_forward_context, FakeAudioVAE）：新增 test_reference_audio_encode_sets_forward_context，用 FakeAudioVAE 复现 PR#35481 的 ForwardContext 回归，是本 PR 携带该修复的验证证据。
- python/sglang/multimodal_gen/test/server/testcase_configs.py（模块 用例配置；类别 test；类型 data-contract；符号 from_dict, optional_float）：ScenarioConfig 新增 load_peak_vram_mb / runtime_peak_vram_mb 可选字段与 optional_float 解析，支撑 per-scenario 显存基线。
- python/sglang/multimodal_gen/test/server/test_server_utils.py（模块 校验工具；类别 test；类型 core-logic；符号 validate_peak_vram）：PerformanceValidator 新增 validate_peak_vram，实现 load/runtime 独立容差校验，是门禁判定逻辑所在。
- python/sglang/multimodal_gen/test/server/perf_baselines/h100.json（模块 H100 基线；类别 test；类型 configuration）：一次性为全部 H100 perf 用例补充 load/runtime 显存基线（含 Ref2VA、realtime、Wan LoRA 等），任何用例缺基线都会触发 fail。
- python/sglang/multimodal_gen/test/unit/test_hf_diffusers_utils.py（模块 下载工具；类别 test；类型 test-coverage；符号 test_cached_snapshot_respects_requested_revision）：新增 revision 下载缓存单测，配套提交 fix(diffusion): honor model revision in pipeline downloads，保证 pinned checkpoint 生效。
- python/sglang/multimodal_gen/test/unit/test_disagg_roles.py（模块 角色配置；类别 test；类型 test-coverage；符号 test_load_config_passes_server_revision_to_model_download）：新增单测验证加载配置会把 server revision 传给模型下载，支撑 Ref2VA checkpoint 固定。

关键符号：_record_replica_peak_memory, _record_output_peak_memory, validate_peak_vram, _validate_and_record, _validate_audio_consistency, compare_audio_with_gt, probe_audio_stream, extract_audio_pcm, save_audio_gt_artifact, decode_audio_gt_wav, minimax_h3_encode_reference_audio_rows

关键源码片段

python/sglang/multimodal_gen/runtime/managers/gpu_worker.py

唯一的生产代码变更：GPUWorker 新增 load/runtime 双显存峰值记录，并通过 replica group all-reduce(MAX) 聚合，是显存门禁的数据源头。

```
class GPUWorker(GPUWorkerPostTrainingMixin):
    def __init__(self, local_rank, rank, master_port, server_args):
        ...
        self.init_device_and_model()

        # 冻结 " 加载期 " 显存峰值：init_device_and_model 内所有 checkpoint
        # 物化、权重搬运的显存占用都被计入 load peak；CPU 平台没有
```

```
        # CUDA allocator 统计，统一以 0 占位，避免非 CUDA 路径误入集合通信。
        self._load_peak_reserved_mb = (
            0.0
            if current_platform.is_cpu()
            else capture_memory_snapshot().peak_reserved_mb
        )
        # " 服务期 " 峰值从零开始累计，只覆盖 warmup 与正式请求，
        # 让 transient checkpoint 物化与 serving-time workspace/activation
        # 回归无法互相掩盖。
        self._runtime_peak_reserved_mb = 0.0
        ...

    def _record_output_peak_memory(self, output_batch: OutputBatch) -> None:
        # 从 " 仅 output rank 记录 " 改为每个 rank 都累计本地 runtime 峰值；
        # 只有 output rank 额外把当前峰值写回 batch，供单卡视角使用。
        if current_platform.is_cpu():
            return
        peak_reserved_mb = capture_memory_snapshot().peak_reserved_mb
        self._runtime_peak_reserved_mb = max(
            self._runtime_peak_reserved_mb, peak_reserved_mb
        )
        if self.is_output_rank:
            output_batch.peak_memory_mb = peak_reserved_mb

    def _record_replica_peak_memory(self, output_metrics: list[Any]) -> None:
        # 注：该函数主体为根据单测契约整理的实现片段（非逐行摘录）。
        # 通过 replica group 的 all-reduce(MAX) 聚合各 rank 的 load/runtime
        # 峰值，保证所有 rank 记录一致的 " 副本最大值 "；单测契约中
        # all_reduce 返回 [5120.0, 3584.0] 时，load_peak 与 runtime_peak
        # 分别写入每个请求的 memory_snapshots。
        replica_group = get_replica_group()
        peaks = torch.tensor(
            [self._load_peak_reserved_mb, self._runtime_peak_reserved_mb],
            dtype=torch.float64,
            device=current_platform.get_device(),
        )
        all_peaks = replica_group.all_reduce(peaks, op=torch.distributed.ReduceOp.MAX)
        load_peak_mb, runtime_peak_mb = all_peaks.tolist()
        for metrics in output_metrics:
            metrics.record_memory_snapshot(
                "load_peak", MemorySnapshot(0.0, 0.0, load_peak_mb, load_peak_mb)
            )
            metrics.record_memory_snapshot(
                "runtime_peak",
                MemorySnapshot(0.0, 0.0, runtime_peak_mb, runtime_peak_mb),
            )
```

python/sglang/multimodal_gen/test/test_utils.py

新增整套音频一致性工具链（探测、PCM 抽取、四指标对比、WAV GT 编解码与 artifact 保存），并把 ci-data revision 升级到携带音频 GT 的提交。

```
@dataclass(frozen=True)
class AudioStreamInfo:
    sample_rate: int
    channels: int
    duration_seconds: float


def probe_audio_stream(file_path: str) -> AudioStreamInfo:
    """返回媒体文件中第一个音频流的元数据。"""
    try:
        result = subprocess.run(
            [
                "ffprobe",
                "-v", "error",
                "-show_entries",
                "stream=codec_type,sample_rate,channels,duration:format=duration",
                "-of", "json",
                file_path,
            ],
            check=True,
            capture_output=True,
            text=True,
        )
    except (FileNotFoundError, subprocess.CalledProcessError) as exc:
        stderr = getattr(exc, "stderr", None)
        raise AssertionError(
            f"Unable to inspect audio stream in {file_path}: {stderr or exc}"
        ) from exc

    payload = json.loads(result.stdout)
    stream = next(
        (
            item
            for item in payload.get("streams", [])
            if item.get("codec_type") == "audio"
        ),
        None,
    )
    # 无音频流、采样率 / 声道 / 时长为 0 都属于 " 不合格输出 ",
    # 统一断言失败而不是静默通过，确保静音 / 无音轨视频被拦截。
    assert stream is not None, f"Media file has no audio stream: {file_path}"

    sample_rate = int(stream.get("sample_rate") or 0)
    channels = int(stream.get("channels") or 0)
    duration = float(
        stream.get("duration") or payload.get("format", {}).get("duration") or 0.0
```

```
    )
    assert sample_rate > 0, f"Audio stream has invalid sample rate: {sample_rate}"
    assert channels > 0, f"Audio stream has invalid channel count: {channels}"
    assert (
        math.isfinite(duration) and duration > 0
    ), f"Audio stream has invalid duration: {duration}"
    return AudioStreamInfo(sample_rate, channels, duration)
```

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

把所有 CUDA perf 用例接入显存峰值门禁，并新增音频一致性校验分支；缺基线时自动 dump 并 fail，是 CI 门禁的落地关键。

```
# _validate_and_record 内新增的显存峰值校验分支：仅在 CUDA 平台生效，
# 任何 perf 用例缺基线都会先落盘基线 JSON 再 fail，
# 保证一次 CI 运行就能给出 " 新基线所需全部数据 "。
if current_platform.is_cuda():
    expected_load_peak_vram_mb = scenario.load_peak_vram_mb
    expected_runtime_peak_vram_mb = scenario.runtime_peak_vram_mb
    if (
        expected_load_peak_vram_mb is None
        or expected_runtime_peak_vram_mb is None
    ):
        self._dump_baseline_for_testcase(case, summary, missing_scenario)
        pytest.fail(
            f"Testcase '{case.id}' is missing a load/runtime peak VRAM "
            f"baseline in {get_perf_baseline_path()}"
        )
    try:
        validator.validate_peak_vram(
            summary,
            expected_load_peak_vram_mb,
            expected_runtime_peak_vram_mb,
        )
    except AssertionError as e:
        logger.error(f"Peak VRAM validation failed for {case.id}:\n{e}")
        self._dump_baseline_for_testcase(case, summary, missing_scenario)
        raise
```

评论区精华

本 PR 无任何 reviewer 评论（comments_count=0、review_comments=0），设计取舍记录在 PR 描述与提交信息中。核心论点有三：(1) 显存峰值必须拆分统计，PR 给出实测数据：合并统计时 request-local 峰值 35,994 MiB、包含加载的 replica 峰值 64,310 MiB，相差近 28 GiB，足以掩盖服务期回归；(2) load 峰值在模型初始化后立即冻结，runtime 峰值从零开始覆盖 warmup+ 正式请求，使两类回归无法互相掩盖；(3) 平台化 GT 策略：H100 CI 才是最终平台 GT 与显存基线来源，Blackwell 输出不作为 H100 基准。提交序列中的 "avoid non-CUDA VRAM collectives"、"scope VRAM baselines to CUDA" 也表明作者在跨平台（CPU/MPS/NPU）分支上做了显式规避。

- load 与 runtime 显存峰值必须分离统计 (design): 采用双峰值方案：load 峰在 init_device_and_model 后立即冻结，runtime 峰从零开始累计 warmup+ 正式请求；两者用独立容差（load 1% + 128 MiB、runtime 2% + 128 MiB）校验。
- 非 CUDA 平台避免显存统计与集合通信 (design): load 峰在 CPU 上记为 0，显存集合通信与基线校验仅 CUDA 生效，避免跨平台崩溃。
- 平台化音频 GT 与 H100 基准策略 (question): 平台目录优先的 GT 候选逻辑生效，Blackwell 输出仅用于本地验证，最终基准以 H100 CI 为准。

风险与影响

- 风险：
  - 核心路径变更：gpu_worker.py 的 _execute_forward_common 是 diffusion 推理主路径，peak-reset 改为全员、_record_output_peak_memory 后移并改变 output rank 语义，既有 output.peak_memory_mb 的取值时机随之变化，可能影响依赖该字段的其他链路（如 do_mem_analysis）。
  - 基线收紧风险：显存校验对所有 CUDA perf 用例生效且容差偏紧（load 1% + 128 MiB、runtime 2% + 128 MiB），h100.json 一次性为全部既有用例补基线，若个别基线标定不准会造成上游 CI 频繁误报；runtime 峰值覆盖 warmup 与请求，GPU 频率、并行度波动都可能触发超限。
  - 跨平台适配：CPU/MPS/NPU 通过 is_cpu() 等条件绕过 allocator 统计与集合通信，但新增的 recorder 与校验逻辑没有 NPU/MPS 专门 CI 验证，存在平台分支遗漏风险。
  - GT 仓库联动：SGL_TEST_FILES_CI_DATA_REVISION 升级影响所有一致性检查（影像与音频），若 ci-data-diffusion 仓库数据不完整或 SHA 过期，会导致其他用例连锁失败。
  - 外部依赖：音频校验依赖 ffmpeg/ffprobe 可用性，CI 镜像缺 ffprobe 时用例直接失败。
  - 影响：对用户侧无可观察的运行时行为变化（改动集中在 CI/ 测试链路与指标记录）；对系统侧，diffusion CI 新增一条 2-GPU Ref2VA E2E 用例与音频一致性门禁，所有 CUDA perf 用例新增 load/runtime 双显存峰值校验，指标进入 pytest 报告、合并 retry artifact 与 dashboard；对团队侧，MiniMax-H3 Ref2VA 从 " 零覆盖 " 变为 " 单测 +E2E 双保险 "，后续新增 diffusion 模型或改动加载 / 推理路径时都必须提供相应显存基线，客观上抬高了 CI 维护成本但显著增强回归防护。
  - 风险标记：核心路径变更，基线收紧风险，跨平台适配，外部 GT 仓库联动

关联脉络

- PR #35481 [Diffusion] Fix MiniMax H3 reference audio forward context: 本 PR 分支直接包含其 forward-context 修复（首个 commit），并新增对应回归单测；#35481 合并后该 diff 会从本 PR 中消失。
- PR #35370 [diffusion] feat: load GGUF transformer checkpoints (MiniMax-H3): 同属 MiniMax-H3 模型支持主线，本 PR 在其基础上补齐 Ref2VA 参考音频路径的回归覆盖。
- PR #35626 [diffusion] fix: keep large vocab tables in host memory under layerwise offload: 同为 diffusion 显存管理方向的改进，与本 PR 的显存峰值守卫共同构成显存回归防线。

- PR #35618 [diffusion] UX: report where a component's weights are: 同为 diffusion 加载期可观测性改进，与 load 峰值冻结同属模型加载诊断链路。
- PR #35615 [diffusion] ci: use canonical residency selector: 同属 diffusion CI 配置演进，后续可复用本 PR 的音频一致性工具与显存校验框架。