PR #25964 完整报告

sgl-project/sglang

[EPD] Cross-request batching for image/audio encoder

合并时间：2026-05-26 15:39

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25964

执行摘要

- 一句话：EPD 编码器跨请求批处理图像 / 音频
- 推荐动作：建议精读 encode_server.py 中的 EncoderScheduler 设计，特别是积累窗口机制和预验证逻辑。值得关注的决策包括 tile 扩展适配、音频采样率修复、环境变量统一声明。如果需要扩展批处理到视频或其他模态，可参考此设计模式。

功能与动机

在 EPD 部署中，编码器服务器以前独立处理每个 /encode HTTP 请求：每个请求一次 HF processor 调用 + 一次 encoder forward + 一次 ZMQ broadcast。在并发流量下，对于每个请求较小的 mm_items，编码器吞吐受启动开销主导，GPU 利用率低。本 PR 在编码器服务器内引入跨请求批处理。

实现拆解

1. 新增 PendingRequest 类和 EncoderScheduler 类。FastAPI lifespan 启动后台批处理 worker，从 asyncio.Queue 拉取待处理的 /encode 请求，按模态分组，分发到 batch_encode。
2. 实现 MMEncoder.batch_encode 方法，将同模态请求的 mm_items 拼接为单一 processor 输入，执行一次 encoder forward，再按 items_per_req 将结果切片回各请求。
3. 添加预广播验证 _validate_request_shape，在 rank 0 广播 batch_encode 任务前检查请求字段合法性，避免因单条无效请求导致 TP 死锁。
4. 修复 Kimi-VL/K25 的 tile 扩展对齐问题：新增 _grid_count_per_leaf 计算每个 leaf 的 grid 数，batch slicing 在 grid 空间进行而非 leaf 空间。
5. 将编码器相关环境变量（SGLANG_ENCODER_IMAGE_PROCESSOR_USE_GPU / MAX_BATCH_SIZE / REQ_TIMEOUT）从分散的 os.getenv 迁移到 environ.py 的 EnvBool/EnvInt/EnvFloat 声明。
6. 修正 MiMo-V2 音频采样率推理：添加 _resolve_audio_sr 从模型配置中读取预期采样率，传递给 load_audio，避免采样率不匹配导致波形扭曲。同时修正音频 hash 计算和 encode-missing 路径，区分音频与图像 / 视频的布局。

关键文件：

- python/sglang/srt/disaggregation/encode_server.py（模块 编码器；类别 source；类型 core-logic；符号 PendingRequest, EncoderScheduler, batch_encode, encode_request）

：核心实现文件，引入跨请求批处理调度器 EncoderScheduler、batch_encode 方法、预验证、tile 扩展修复、音频采样率推理等所有核心逻辑。

- python/sglang/srt/multimodal/processors/mimo_v2.py（模块 处理器；类别 source；类型 dependency-wiring；符号 init）：将图像处理器 GPU 标志从 os.getenv 切换为 envs 访问，统一环境变量管理。
- python/sglang/srt/environ.py（模块 环境配置；类别 source；类型 configuration）：声明编码器批处理相关的三个环境变量，统一配置声明。

关键符号：PendingRequest, EncoderScheduler, batch_encode, _validate_request_shape, _grid_count_per_leaf, _resolve_audio_sr

关键源码片段

python/sglang/srt/disaggregation/encode_server.py

核心实现文件，引入跨请求批处理调度器 EncoderScheduler、batch_encode 方法、预验证、tile 扩展修复、音频采样率推理等所有核心逻辑。

```
# encode_server.py — 跨请求批处理核心类

import asyncio
import time
from collections import defaultdict
from concurrent.futures import ThreadPoolExecutor

from sglang.srt.environ import envs
from sglang.srt.managers.schedule_batch import Modality

# 从 typed envs 读取编码器批处理配置
ENCODER_MAX_BATCH_SIZE = envs.SGLANG_ENCODER_MAX_BATCH_SIZE.get()
ENCODER_REQ_TIMEOUT = envs.SGLANG_ENCODER_REQ_TIMEOUT.get()


class PendingRequest:
    """包装待处理编码请求及其 asyncio.Future"""
    __slots__ = ("request", "future", "submit_time")

    def __init__(self, request: dict, loop: asyncio.AbstractEventLoop):
        self.request = request
        self.future: asyncio.Future = loop.create_future()
        self.submit_time = time.time()


# 可批处理的模态（图像 / 音频），视频因预处理参数多变被排除
_BATCHABLE_MODALITIES = {Modality.IMAGE, Modality.AUDIO}


class EncoderScheduler:
    """跨请求编码批处理调度器"""
```

```
def __init__(self, pending_queue: asyncio.Queue, mm_encoder: "MMEncoder"):
    self.pending_queue = pending_queue
    self.mm_encoder = mm_encoder
    self.io_executor = ThreadPoolExecutor(
        max_workers=envs.SGLANG_ENCODER_MM_LOAD_WORKERS.get()
    )

async def _collect_batch(self) -> list:
    """收集最多 ENCODER_MAX_BATCH_SIZE 个请求"""
    batch = [await self.pending_queue.get()] # 第一个会等待
    while len(batch) < ENCODER_MAX_BATCH_SIZE:
        try:
            batch.append(self.pending_queue.get_nowait())
        except asyncio.QueueEmpty:
            break
    return batch

async def run_batch_worker(self):
    """后台主循环：收集批次、按模态分组、分发"""
    while True:
        batch = await self._collect_batch()
        if not batch:
            continue
        groups = defaultdict(list)
        for pr in batch:
            mod = Modality.from_str(pr.request.get("modality", ""))
            if mod in _BATCHABLE_MODALITIES:
                groups[mod].append(pr)
            else:
                # 视频等不可批处理请求回退到单请求路径
                asyncio.ensure_future(self._process_single(pr))
        for mod, group in groups.items():
            asyncio.ensure_future(self._dispatch_group(group, mod))
```

评论区精华

- ZhengWG 询问批处理积累窗口是否有效，Abatom 解释由于前一批下载时事件循环交出控制权，新请求能够积累，批处理大小通常超过 1。
- ZhengWG 指出 Kimi-VL/K25 的 tile 扩展会破坏批处理切片，Abatom 添加了 _grid_count_per_leaf 在 grid 空间切片。
- ShangmingCai 建议广播前验证批处理是否可处理，Abatom 添加了 _validate_request_shape。
- gemini-code-assist[bot] 指出 aux_data 可能带整个批次的数据给每个请求，Abatom 移除了 batch_encode 中的 aux_data 构建（因为 batch_encode 只处理 IMAGE/AUDIO，不涉及 video-meta）。
- ZhengWG 建议将环境变量集中到 environ.py，完成。

- 批处理积累窗口仅取 1 个请求 (performance): Abatom 解释前一批下载时事件循环释放，新请求可积累，批次大小通常大于 1。
- Kimi-VL tile 扩展破坏切片 (correctness): Abatom 添加 _grid_count_per_leaf 返回每个 leaf 的 grid 计数，切片在 grid 空间进行。
- 广播前验证批次有效性 (correctness): Abatom 添加 _validate_request_shape 进行预验证，拒绝无效请求避免 TP 死锁。
- 批次级 aux_data 透传给各请求 (correctness): Abatom 移除了 batch_encode 中的 aux_data 构建（因为 IMAGE/AUDIO 不涉及其内容）。
- 环境变量集中到 environ.py (refactor): 完成迁移。

风险与影响

- 风险：
  - 超时机制（SGLANG_ENCODER_REQ_TIMEOUT）若设置过短，可能在高负载下返回 504；若 NCCL 真正 hang，超时前仍会阻塞。
  - 批次内任何一个请求在 processor-time 出错会导致整个批次失败（目前有预验证隔离结构性问题，但 processor 内错误仍会广播失败）。
  - 移除了 mm_global_cache 路径，依赖该缓存的用户会退化为无缓存路径。
  - 音频采样率回退到 16000 Hz 若不符合模型预期，将导致波形失真。
- 影响：
  - 用户：批处理带来吞吐提升，但个别请求可能因积累延迟增加；超时返回 504，需客户端容错。
  - 系统：减少编码器启动开销，提高 GPU 利用率；新增三个环境变量默认值安全，可配置。
  - 团队：核心代码集中在 encode_server.py，易于扩展其他模态批处理；需关注批次失败处理和超时配置。
  - 风险标记：批次失败影响所有请求，音频采样率回退可能失真，全局缓存路径被移除，超时可能掩盖 NCCL 死锁

关联脉络

- PR #26281 [CI] Enable EPD CI for EPD architecture enhancements: 构架 EPD CI 测试，本 PR 实现了编码器批处理，两者均属于 EPD 架构演进。