# PR #30398 完整报告

sgl-project/sglang

[Refactor] New EPD

合并时间：2026-08-21 15:22

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30398

## 执行摘要

- 一句话：重构 EPD encoder 为分层包，统一 encode/send 生命周期
- 推荐动作：建议精读。这个 PR 是 EPD 编码器从单文件到分层包的关键转折点，值得关注的设计决策包括：用单一生命周期消灭重复编排、用 ReqState 显式表达资源所有权、把传输差异压缩到 delivery 抽象与三个抽象方法、以及接收端池化的 FIFO/ 无 TTL 取舍。对后续想接入 gRPC 或 Rust 编码器的工程师尤其有参考价值。

## 功能与动机

PR body 明确指出原 encode_server.py 已膨胀到 4464 行，同时拥有 FastAPI 应用、DP launcher、请求调度、CPU 预处理、GPU forward 和两个传输后端，带来三个具体问题：一是 HTTP 非 DP 与 DP worker 各自维护一份编排逻辑，单发 / 批发、Mooncake/ZMQ 三套 send 逻辑高度重复；二是 send 隐式释放 embedding 且清理分散在正常 / 错误 / 超时分支，release 与 in-flight forward 存在竞态，失败 encode 可能让 send 永久等待；三是没有协议中立层，gRPC 无法复用 HTTP 后端拓扑，CPU 预处理和 HTTP 层也无法被 Rust 实现替换。

## 实现拆解

1. 目录与职责分层：旧 python/sglang/srt/disaggregation/encode_server.py（-4602 行）被删除，新包 python/sglang/srt/disaggregation/encoder/ 按层拆分：server.py 保留 MMEncoder、ReqState、EncoderDelivery 与 EncoderMetaRegistry 等 GPU 侧核心；runtime.py 托管 EncoderScheduler、DP worker 进程与 execute_encode_pipeline；preprocessor.py 承载 CPU 预处理；receiver.py 从旧 encode_receiver.py 搬迁并抽象接收端；http_server.py 与 grpc_server.py 只保留各协议入口，原 encode_grpc_server.py 同步改名。
2. 统一请求生命周期：receive 之后所有路径（HTTP 非 DP、DP worker）统一走 execute_encode_pipeline。encode() 退化为 batch_encode() 的 batch-of-one 调用，调度器只决定是否合并；ReqState 以 _acquire_encode_ref() / _release_encode_ref() 维护 active_encodes、active_sends、release_requested 等状态，保证 release 不会越过未完成 send，失败 encode 也会 stage 错误，send 不会永久阻塞。scheduler.py 与 tokenizer_manager.py 的导入随包结构同步调整。
3. 传输后端收敛：原 ZmqSchedulerDelivery 与 ZmqTokenizerDelivery 合并为按 cleanup_receive_state 参数化的 ZmqDelivery；Mooncake 与 ZMQ 的各类清理统一收口到 release_request()；transfer_backend / use_mooncake 缓存在 MMEncoder 上，不再

到处读 server_args。Mooncake 侧 embedding 只有在 CUDA stream 同步后才标记 ready，避免 RDMA 读到尚未落盘的 GPU buffer。

4. 预处理元数据集中：EncoderPreprocessResult 同时返回 mm_inputs、grid_thw 与 token_counts；grid 归一化、patch/token 计数从 MMEncoder 移入 EncoderPreprocessor，process_mm_items 只解析一次 grid，direct/cache/Mooncake/batch 复用同一份结果，slice_embedding 不再重复计算。
5. 接收端池化与配套测试：EmbeddingPool 吸收原 MooncakeEmbeddingPool，新增 try_stage / release_on_gc；显式设置 SGLANG_EMBEDDING_POOL_SIZE_MB 时 zmq_to_scheduler 也启用 GPU 池（默认 4096 MB 仍只作用于 Mooncake，默认行为不变）。WaitingMMRequestBase（ABC）统一 recv_embedding，后端差异缩小到三个抽象方法。测试更新集中在 test/registered/unit/disaggregation/ 下的 test_encode_server.py、test_encoder_health.py、test_kimi_k3_encoder_mode.py，覆盖 delivery 契约、cleanup_receive_state 可配置性与 Kimi grid 归一化。

关键文件：

- python/sglang/srt/disaggregation/encoder/server.py（模块 编码核心；类别 source；类型 core-logic；符号 MMEncoder, ReqState, EncoderDelivery, ZmqDelivery）：新包的核心：保留 MMEncoder、ReqState、EncoderDelivery、EncoderMetaRegistry 等 GPU 侧逻辑，承接原 encode_server.py 的主体，并实现 receive 汇合与健康检查。
- python/sglang/srt/disaggregation/encoder/runtime.py（模块 编码调度；类别 source；类型 core-logic；符号 PendingRequest, EncoderScheduler, _resolve_encoder_batch_policy, execute_encode_pipeline）：协议无关 runtime 与 EncoderScheduler 的落点，execute_encode_pipeline 统一生命周期、DP worker 托管都在这里。
- python/sglang/srt/disaggregation/encoder/receiver.py（模块 接收端；类别 source；类型 rename-or-move；符号 EmbeddingData, EmbeddingPool, WaitingMMRequestBase, WaitingImageRequest）：从 encode_receiver.py 重命名并抽象：WaitingMMRequestBase 统一 recv_embedding，EmbeddingPool 吸收 MooncakeEmbeddingPool 并对 ZMQ 路径开放 try_stage。
- python/sglang/srt/disaggregation/encoder/preprocessor.py（模块 预处理器；类别 source；类型 core-logic；符号 EncoderPreprocessor, EncoderPreprocessResult, _convert, _get_original_image_size）：CPU 预处理独立成模块，EncoderPreprocessResult 统一 mm_inputs/grid_thw/token_counts，四个编码路径复用同一份解析结果。
- python/sglang/srt/disaggregation/encoder/http_server.py（模块 服务入口；类别 source；类型 entrypoint；符号 launch_server, _lifespan, _register_encoder_url_with_bootstrap, _try_register_once）：HTTP 层瘦身为 FastAPI 路由与生命周期，业务编排委托给 runtime；明确设计为可被 Rust 替换的薄层。
- python/sglang/srt/disaggregation/encode_server.py（模块 旧实现；类别 source；类型 deletion）：4464 行旧单文件实现被删除，确认重构不是原地修改而是真正的分层替换。
- python/sglang/srt/disaggregation/encoder/grpc_server.py（模块 远程接口；类别 source；类型 rename-or-move）：随包重构从 encode_grpc_server.py 改名搬迁，保持

gRPC 入口兼容。

- test/registered/unit/disaggregation/test_encode_server.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestEncoderPreprocessorKimiGrid, TestEncoderDelivery, test_contract_has_two_direct_implementations, test_zmq_delivery_cleanup_is_configurable）：测试适配新包结构，并新增 delivery 契约与 ZMQ cleanup 可配置性测试。

关键符号：*execute_encode_pipeline*, batch_encode, *encode*, *release_request*, *_acquire_encode_ref*, *_release_encode_ref*, try_stage, *release_on_gc*, *_resolve_encoder_batch_policy*, *_collect_batch*, publish, *_get_mm_grid_dim*, *_aggregate_embedding_part*, *_mark_keep_device_embedding*, *recv_embedding*, *send_encode_request*

## 关键源码片段

### python/sglang/srt/disaggregation/encoder/server.py

新包的核心：保留 MMEncoder、ReqState、EncoderDelivery、EncoderMetaRegistry 等 GPU 侧逻辑，承接原 *encode_server.py* 的主体，并实现 *receive* 汇合与健康检查。

```
# EPD 编码器模块级的接收汇合状态:
# HTTP 与 DP 入口在收到同一个 req_id 的多个分片时先在此登记,
# 各分片到达后通过条件变量唤醒等待方（对应 receive 阶段）。
rid_lock = asyncio.Lock()
rid_to_receive_endpoint = dict() # req_id -> 已登记的 receive endpoint 集合
rid_to_receive_count = dict() # req_id -> 期望接收的分片数
rid_to_cond = {} # req_id -> asyncio.Condition


async def _get_receive_condition(req_id: str) -> asyncio.Condition:
    # 每个 req_id 只创建一次 Condition，后续分片复用同一个等待点
    async with cond_dict_lock:
        if req_id not in rid_to_cond:
            rid_to_cond[req_id] = asyncio.Condition()
        return rid_to_cond[req_id]


def is_health_check_request(rid: Optional[str]) -> bool:
    # 健康检查复用普通 encode 管线，但以特殊的 RID 前缀区分,
    # 避免为健康检查单独维护一套路径
    return isinstance(rid, str) and rid.startswith(HEALTH_CHECK_RID_PREFIX)
```

### python/sglang/srt/disaggregation/encoder/runtime.py

协议无关 runtime 与 EncoderScheduler 的落点，*execute_encode_pipeline* 统一生命周期、DP worker 托管都在这里。

```
# VIDEO 由于每请求的预处理参数（do_sample_frames、video_metadata）不同,
# 无法合并进同一个 HF processor 调用，因此只有 IMAGE/AUDIO 可参与 batch。
_BATCHABLE_MODALITIES = {Modality.IMAGE, Modality.AUDIO}
```

```
# Kimi K3 未显式配置时的默认编码 batch 上限。
_KIMI_K3_DEFAULT_ENCODER_MAX_BATCH_SIZE = 2


def _resolve_encoder_batch_policy(
    model_type: str,
    configured_max_batch_size: int,
    max_batch_size_is_explicit: bool,
) -> Tuple[int, bool]:
    """返回生效的 max_batch_size 与是否启用同轮合并（coalesce_same_turn）。"""
    max_batch_size = max(1, int(configured_max_batch_size))
    # 当前只有 Kimi K3 默认做同轮合并；未显式设置上限时自动收窄，
    # 避免默认 batch 过大导致显存或延迟失控。
    coalesce_same_turn = model_type == "kimi_k3"
    if coalesce_same_turn and not max_batch_size_is_explicit:
        max_batch_size = min(
            max_batch_size, _KIMI_K3_DEFAULT_ENCODER_MAX_BATCH_SIZE
        )
    return max_batch_size, coalesce_same_turn


class EncoderScheduler:
    """把并发的 /encode 请求聚合成有界 batch，只决定是否合并，
    不再维护第二套单请求 encode 实现（encode 是 batch_encode 的特例）。"""

    def __init__(
        self,
        encoder: "MMEncoder",
        send_sockets: List[zmq.Socket],
        max_batch_size: int,
        coalesce_same_turn: bool = False,
        request_timeout: float = server_module.ENCODER_REQ_TIMEOUT,
    ):
        self.encoder = encoder
        self.send_sockets = send_sockets
        self.max_batch_size = max(1, int(max_batch_size))
        self.coalesce_same_turn = bool(coalesce_same_turn)
        self.request_timeout = max(1.0, float(request_timeout))
        self.pending_queue: asyncio.Queue["PendingRequest"] = asyncio.Queue()
        self._worker_task: Optional[asyncio.Task] = None
```

## python/sglang/srt/disaggregation/encoder/receiver.py

从 encode_receiver.py 重命名并抽象：WaitingMMRequestBase 统一 recv_embedding，EmbeddingPool 吸收 MooncakeEmbeddingPool 并对 ZMQ 路径开放 try_stage。

```
def _mark_keep_device_embedding(mm_inputs) -> None:
    """要求 general_mm_embed_routine 不要把 embedding 拷回 CPU，
    因为接收端可能会以零拷贝视图直接消费 GPU 上的 embedding。"""
    if mm_inputs is None:
```

```
        return
    for item in mm_inputs.mm_items:
        item.keep_device_embedding = True


class EmbeddingData:
    def __init__(
        self,
        req_id,
        num_parts,
        part_idx,
        grid_dim,
        modality,
        embedding=None,
        embedding_shape=None,
        error_msg=None,
        error_code=None,
        **kwargs,
    ):
        self.req_id = req_id
        self.num_parts = num_parts
        self.part_idx = part_idx
        self.grid_dim = grid_dim
        self.modality = modality
        self.embedding = embedding
        self.send_time = None
        self.dtype = embedding.dtype if embedding is not None else None
        if embedding_shape is not None:
            self.shape = embedding_shape
        else:
            self.shape = list(embedding.shape) if embedding is not None else None
        # Encoder 侧 Mooncake MR 地址；下划线前缀让 copy_without_embedding
        # 在跨进程 pickle 时丢掉这个进程本地地址。
        self._mr_ptr: Optional[int] = None
        self.error_msg = error_msg


def _aggregate_embedding_part(current, recv_obj, model_type):
    """把接收到的某一个分片折叠进聚合结果（第一个分片负责创建）。"""
    if current is None:
        return MultiModalEmbeddingData.from_embedding_data(
            recv_obj, model_type=model_type
        )
    current.add(recv_obj)
    return current
```

## 评论区精华

- 结构要求：ShangmingCai 要求把所有 encoder 文件移入 disaggregation/encoder 目录并去掉前缀，同时指出这是相当大的重构，需要清晰的 PR 描述和测试结果。
- HTTP 层错误码：ShangmingCai 在 server.py 上建议「既然这是 HTTP 可触达的位置，条件不满足时是否应该返回 BadRequestError 而不是断言」，作者回复 done，已处理。
- 大请求饥饿：ShangmingCai 问 receiver.py 里 EmbeddingPool 的实现「大请求会被饿死吗」。作者回应 try_stage() 池满时走 FIFO 队列而非随机淘汰，大请求只会等待空闲槽位、延迟升高而不会饥饿，并依赖调度器背压防止无限排队。
- 槽位 TTL：ShangmingCai 问池子是否有 TTL；ZhengWG 说明每个终态路径（成功 / 中止 / 失败 / 超时 /stuck pending）都会确定性释放槽位，加墙钟 TTL 反而可能在 mm_inputs 仍引用显存时强制回收，造成静默数据损坏。
- 结论：ZhengWG LGTM，ShangmingCai 最终 APPROVED 并留言「No other comment, looks good.」
- HTTP 可触达位置应返回 BadRequestError 而非断言 (design)：作者回复 done，已改为 HTTP 友好的错误返回。
- EmbeddingPool 池满时大请求是否会饥饿 (performance)：接受现状，无后续代码改动。
- EmbeddingPool 槽位是否需要显式 TTL (correctness)：不加 TTL，作为 follow-up 话题保留。
- 大规模重构需要 PR 描述与测试结果 (documentation)：PR body 补充了分层说明，cccccya 随后贴出 Qwen3-VL 单发 / 批发及 MM 全局缓存性能对比。

## 风险与影响

- 风险：
  - 核心路径回归：旧 encode_server.py 被整体删除，receiver.py 发生 rename，scheduler.py / tokenizer_manager.py 的导入同步调整；任何遗漏的符号迁移都会在启动期失败。合并过程中多次 main 冲突也说明长期分支的集成风险。
  - 并发生命周期正确性：ReqState 依赖 _acquire_encode_ref() / _release_encode_ref() 与 release_request() 的完整性；若某个终态分支漏调 release，可能出现槽位泄漏或 send 永久等待。
  - 饥饿与延迟：池满时 try_stage() 采用 FIFO，大请求等待时间可能升高；目前依赖调度器背压控制排队深度，缺少显式 TTL。
  - Mooncake 同步：若 CUDA stream 同步顺序被破坏，RDMA 可能读到未完成的 GPU embedding，属于隐蔽的数据正确性风险。
  - HTTP 可触达路径错误码：review 指出需要把断言改为 BadRequestError；若回归，用户可能拿到 500 而不是 400。
- 影响：
  - 用户 /API：对外 HTTP 与 gRPC surface 保持不变，请求语义不变，EPD 用户无迁移成本。
  - 系统行为：单发与批发统一实现，receive 后所有后端共享同一生命周期，错误与超时路径行为趋于一致；性能压测显示与 main 持平。

- 团队 / 架构：新增协议中立层，gRPC 可复用 HTTP 的后端拓扑；preprocessor.py 与 http_server.py 明确设计为可被 Rust 替代，为后续高性能实现铺路。
- 风险标记：核心路径重构，并发资源生命周期，池满时大请求延迟，HTTP 可触达校验错误码，Mooncake CUDA 同步正确性

## 关联脉络

- PR #27770 [P/D disagg] Decode-side radix cache for SWA hybrid models (unified radix tree): 同属分离式推理（disaggregation）基础设施演进，与本 PR 中 EPD 编码侧接收 / 缓存重构共同完善 P/D 全链路。
- PR #35718 Support mxfp8 KV cache in PD transfer: 同为 PD 分离式传输能力建设，与本 PR 中 Mooncake/ZMQ 传输与 embedding 落地路径的抽象方向互补。