PR #23049 完整报告

sgl-project/sglang

[Diffusion] Diffusion model support log-requests

合并时间：2026-07-05 17:01

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23049

执行摘要

- 一句话：为 Diffusion 服务添加请求日志记录功能
- 推荐动作：值得快速浏览，了解如何复用 srt 的日志基础设施为 diffusion 路径添加可观测性。重点关注 DiffusionRequestLogger 的设计（采样配置白名单、级别语义）以及钩子插入点（scheduler_client.forward）。该 PR 为后续 diffusion 服务的运维监控奠定了基础。

功能与动机

来自 PR body 的表述：'Diffusion model serving (image/video generation) did not support --log-requests, --log-requests-level, --log-requests-format, or --log-requests-target, so these flags were rejected at startup.' 此 PR 将这些功能引入 diffusion 运行时，满足用户对 diffusion 服务请求级日志的需求。

实现拆解

1. 新建 DiffusionRequestLogger 类（python/sglang/multimodal_gen/runtime/utils/request_logger.py）：该类镜像 srt 的 RequestLogger 设计，定义了 10 个采样配置白名单字段（_SAMPLING_CONFIG_FIELDS），并通过log_requests_level控制日志详略（0-3级）。通过 from_server_args 工厂方法从 ServerArgs 初始化，底层使用 srt 的 create_log_targets 支持多目标（stdout 和文件）输出。
2. 添加 CLI 参数（python/sglang/multimodal_gen/runtime/server_args.py）：在 ServerArgsdataclass中新增log_requests、log_requests_level、log_requests_format、log_requests_target 四个字段，并在 add_cli_args 中注册对应的 argparse 参数。
3. 在 scheduler_client 中插入日志钩子（python/sglang/multimodal_gen/runtime/scheduler_client.py）：在 SchedulerClient.initialize 中创建 DiffusionRequestLogger 实例；在 forward() 方法开始前调用 log_received_request(batch)，在成功收到响应后调用 log_finished_request(batch, output_batch)。AsyncSchedulerClient 类也做了相同的改造，保证同步和异步路径的一致性。
4. 编写集成测试（python/sglang/multimodal_gen/test/server/test_request_logger.py）：使用 ServerManager 启动带有不同日志格式（text/json）的 diffusion 服务，分别测试图像和视频生成场景，验证 stdout 和文件两种目标是否正确输出 request.received 和 request.finished 事件。

5. 更新 CLI 文档（docs_new/docs/sglang-diffusion/api/cli.mdx）：在 Common Options 下新增 "Request logging" 小节，说明四个参数的用法和默认值。

关键文件：

- python/sglang/multimodal_gen/runtime/utils/request_logger.py（模块 日志记录器；类别 source；类型 core-logic；符号 DiffusionRequestLogger, init, from_server_args, _request_id）：核心新增文件，实现 DiffusionRequestLogger 类，定义了日志级别、采样配置白名单、接收 / 完成记录的格式化逻辑。
- python/sglang/multimodal_gen/runtime/scheduler_client.py（模块 调度客户端；类别 source；类型 dependency-wiring；符号 SchedulerClient.initialize, SchedulerClient.forward, AsyncSchedulerClient.initialize, AsyncSchedulerClient.forward）：日志钩子插入点，在 SchedulerClient 和 AsyncSchedulerClient 的 forward() 中调用日志记录，是日志功能与调度路径结合的关键位置。
- python/sglang/multimodal_gen/runtime/server_args.py（模块 服务参数；类别 source；类型 dependency-wiring；符号 log_requests, log_requests_level, log_requests_format, log_requests_target）：添加了四个日志相关的 CLI 参数定义和默认值，是用户启用日志的配置入口。
- python/sglang/multimodal_gen/test/server/test_request_logger.py（模块 集成测试；类别 test；类型 test-coverage；符号 _start_server, _cleanup_server, _create_client, _wait_for_video_completion）：集成测试，验证日志输出到 stdout 和文件，覆盖 text/json 格式和图像 / 视频模型。
- docs_new/docs/sglang-diffusion/api/cli.mdx（模块 文档；类别 other；类型 entrypoint）：更新 CLI 文档，新增 "Request logging" 小节，方便用户了解日志参数。

关键符号：DiffusionRequestLogger.init, DiffusionRequestLogger.from_server_args, DiffusionRequestLogger.log_received_request, DiffusionRequestLogger.log_finished_request, SchedulerClient.forward, AsyncSchedulerClient.forward

关键源码片段

python/sglang/multimodal_gen/runtime/utils/request_logger.py

核心新增文件，实现 DiffusionRequestLogger 类，定义了日志级别、采样配置白名单、接收 / 完成记录的格式化逻辑。

```
# python/sglang/multimodal_gen/runtime/utils/request_logger.py ( 关键部分 )

from typing import Any, Optional
from sglang.srt.utils.log_utils import create_log_targets
from sglang.srt.utils.request_logger import (
    _dataclass_to_string_truncated,
    _transform_data_for_logging,
)

# 记录到日志中的采样配置白名单字段（不含 prompt）
```

```
_SAMPLING_CONFIG_FIELDS = (
    "data_type", "seed", "num_inference_steps", "guidance_scale",
    "true_cfg_scale", "width", "height", "num_frames", "fps",
    "num_outputs_per_prompt",
)

class DiffusionRequestLogger:
    def __init__(self, log_requests: bool, log_requests_level: int,
                 log_requests_format: str, log_requests_target: Optional[list]):
        self.log_requests = log_requests
        self.log_requests_level = log_requests_level
        self.log_requests_format = log_requests_format
        self.log_requests_target = log_requests_target
        # 使用 srt 的 create_log_targets 支持 stdout 和文件多目标
        self.targets = create_log_targets(
            targets=log_requests_target, name_prefix=__name__
        )
        self.log_exceeded_ms = envs.SGLANG_LOG_REQUEST_EXCEEDED_MS.get()
        # level 2 时截断 prompt 为 2 KiB，否则不限制
        self._max_length = (
            _TRUNCATE_LENGTH if self.log_requests_level == 2 else _UNLIMITED
        )

    @classmethod
    def from_server_args(cls, server_args: Any) -> "DiffusionRequestLogger":
        """从 ServerArgs 构造日志记录器实例"""
        return cls(
            log_requests=server_args.log_requests,
            log_requests_level=server_args.log_requests_level,
            log_requests_format=server_args.log_requests_format,
            log_requests_target=server_args.log_requests_target,
        )

    def log_received_request(self, reqs: list) -> None:
        """记录请求接收事件，为每个 req 生成一条日志"""
        if not self.log_requests:
            return
        for req in reqs:
            rid, obj = self._batch_record([req]) # 单请求处理
            self._write_event("request.received", rid, obj)

    def log_finished_request(self, reqs: list, result: Any) -> None:
        """记录请求完成事件，包含 e2e 延迟和可能的错误"""
        if not self.log_requests:
            return
        for req in reqs:
            rid, obj = self._batch_record([req])
            result_view = self._result_view(result)
            self._write_event("request.finished", rid, obj, extra=result_view)
```

```
    def _config_view(self, req: Any, *, drop_seed: bool = False) -> dict:
        """根据日志级别生成配置视图，低于 level 1 返回空"""
        sp = getattr(req, "sampling_params", None)
        if sp is None or self.log_requests_level < 1:
            return {}
        cfg = {name: getattr(sp, name, None) for name in _SAMPLING_CONFIG_FIELDS}
        if drop_seed:
            cfg.pop("seed", None)
        view: dict = {"sampling_params": cfg}
        if self.log_requests_level >= 2:
            view["prompt"] = getattr(sp, "prompt", None)
            view["negative_prompt"] = getattr(sp, "negative_prompt", None)
        return view

    def _result_view(self, result: Any) -> dict:
        """从 result 提取 e2e 延迟和错误信息"""
        e2e_latency = 0.0
        metrics = getattr(result, "metrics", None) if result is not None else None
        if metrics is not None:
            e2e_latency = getattr(metrics, "total_duration_s", 0.0) or 0.0
        return {
            "meta_info": {"e2e_latency": e2e_latency},
            "error": getattr(result, "error", None) if result is not None else None,
        }
```

python/sglang/multimodal_gen/runtime/scheduler_client.py

日志钩子插入点，在 SchedulerClient 和 AsyncSchedulerClient 的 forward() 中调用日志记录，是日志功能与调度路径结合的关键位置。

```
# python/sglang/multimodal_gen/runtime/scheduler_client.py ( 关键改动 )

from sglang.multimodal_gen.runtime.utils.request_logger import (
    DiffusionRequestLogger,
)

class SchedulerClient:
    def __init__(self):
        self.context = None
        self.scheduler_socket = None
        self.server_args = None
        self.request_logger: Optional[DiffusionRequestLogger] = None # 新增 logger 属性

    def initialize(self, server_args: ServerArgs):
        self.server_args = server_args
        # 从 server_args 创建日志记录器（默认不启用）
        self.request_logger = DiffusionRequestLogger.from_server_args(server_args)
        self.context = zmq.Context()
        self.scheduler_socket = self.context.socket(zmq.REQ)
        # ... 原有初始化代码 ...
```

```
    def forward(self, batch: Any, timeout_ms: int | None = None) -> Any:
        """发送请求并等待响应，前后记录日志"""
        # 记录请求进入（scheduler 处理前）
        self.request_logger.log_received_request(batch)
        previous_timeout_ms = None
        if timeout_ms is not None:
            previous_timeout_ms = self.scheduler_socket.getsockopt(zmq.RCVTIMEO)
            self.scheduler_socket.setsockopt(zmq.RCVTIMEO, timeout_ms)
        try:
            self.scheduler_socket.send_pyobj(batch)
            output_batch = self.scheduler_socket.recv_pyobj()
            _materialize_output_batch_file_refs(output_batch)
            # 记录请求完成（收到 scheduler 响应后）
            self.request_logger.log_finished_request(batch, output_batch)
            return output_batch
        except zmq.error.Again:
            logger.error("Timeout waiting for response from scheduler.")
            raise TimeoutError("Scheduler did not respond in time.")
        finally:
            if previous_timeout_ms is not None and self.scheduler_socket is not None:
                self.scheduler_socket.setsockopt(zmq.RCVTIMEO, previous_timeout_ms)
```

评论区精华

Review 中核心讨论包括：

- 测试文件位置争议：mickqian 指出初始测试文件位于 test/registered/utils/test_request_logger.py，要求迁至 python/sglang/multimodal_gen/test。最终 PR 将测试文件置于正确位置，该争议已解决。
- --log-requests 默认值冗余：ping1jing2 指出 action='store_true' 应隐含 default=false，无需显式设置。但最终代码中仍保留了 default=ServerArgs.log_requests，该评论未导致修改。
- 日志应循环所有请求：gemini-code-assist 建议在 scheduler 的事件循环中遍历所有 reqs 记录完成日志，而不仅是第一条。当前实现中 log_finished_request 本身已对列表循环，但在 scheduler.py 的事件循环里仍只调用了一次。该建议未确认采纳，属于潜在的正确性关注点。
  - Diffusion 测试文件应移至正确位置 (testing): 采纳，最终测试文件移至 python/sglang/multimodal_gen/test/server/test_request_logger.py。
  - --log-requests 使用 action='store_true' 时默认值冗余 (style): 未修改，最终代码仍保留了 default=ServerArgs.log_requests。
  - log_finished_request 应循环所有请求 (correctness): 未明确采纳，但 PR 最终代码在 scheduler_client.forward 中已对 batch 列表循环，而 scheduler.py 中的调用点未修改。

风险与影响

- 风险：

1. 性能开销：当 --log-requests-level >= 2 时会序列化提示词（最大 2 KiB），高频请求下可能增加 IPC 和 I/O 负载。但日志默认关闭（False），用户需主动启用。
2. 敏感信息泄露：提示词可能包含用户敏感信息，日志文件若未妥善保护可能造成泄露。默认仅记录元数据，级别 3 才输出完整提示。
3. 调度器路径侵入：在 SchedulerClient.forward() 中增加日志钩子，若 logger 初始化失败可能阻塞请求路径。当前 logger 初始化不依赖外部资源，风险较低。
4. 测试覆盖：集成测试覆盖了 text/json 格式和图像 / 视频模型，但未覆盖异步路径和错误场景（如超时、目标目录不可写）。 - 影响：用户：现在可以通过指定 --log-requests 等参数为 diffusion 服务启用请求级日志，便于调试、监控和审计。日志格式与 LLM 路径一致，降低了运维学习成本。 系统：新增的日志输出会增加磁盘 / 网络 I/O，但默认关闭，按需启用。MTP 等辅助输出不受影响。 团队：维护一个独立的 DiffusionRequestLogger，与 SRT 的 RequestLogger 代码路径平行，后续功能演进需要同步更新。

- 风险标记：日志性能开销，敏感信息泄露风险，调度器路径侵入

关联脉络

- PR #30107 [diffusion] perf: add unified SP shard helpers and zero-copy tail-pad attention: 同属 diffusion 模块的基础设施改进，共享部分运行时依赖（如 scheduler_client.py 等）。该 PR 的日志功能与 SP 重构都是 diffusion 可观测性与性能演进的一部分。
- PR #30110 [diffusion] fix: shut down diffusion workers on serve exit: 同为 diffusion 运行时改进，涉及服务生命周期管理，与请求日志均属于服务可运维性提升。