v0.3 • 2025-08-01

Overview / 摘要

本项目将 MedSAM2 集成到 MONAI Label 服务器，实现与 3D Slicer 等客户端的交互式 3D 医学图像分割。功能包括自定义样本选择策略、面向前端监控的 TCP 日志，以及基于提示的推理管线（多点 3D 提示、前/后向传播、交集与后处理）。

App **Entrypoint**
`main.py` returns app instance

Inference **MedSAM2**
Prompt-aware 3D pipeline

UI **TCP Logs**
Live log streaming to front end

Key Features / 主要特性

- 提供 MONAI Label 应用入口与自定义策略注册。
- 基于数据仓库（datastore_v2.json）的研究/图像元数据管理。
- 前端可选择具体图像（非仅随机采样）。
- 基于 MedSAM2 的提示感知推理与后处理管线。
- 支持 TCP 日志监控并推送至前端界面。

Directory Structure / 目录结构

Copy

```
main.py                 - MONAI Label app entrypoint, returns the app instance
SAM2_MONAI.py           - App config: integrates SAM-Med3D inference & selection strategy
medsam2_monai_utils/    - Utilities: IO, preprocessing, prompts, inference, postprocessing
start_server.sh         - One-click script to start the MONAI Label server
tcp_client.py           - TCP logger client to stream logs to front end
```

Dependencies / 依赖

- torch, torchvision
- numpy, nibabel
- monai-label
- torchio *（可能需要降级到 <0.3）*
- medim

Configuration / 配置

MONAI_LABEL_DATASTORE： 设置为 studies 根目录（图像存放位置）。

Quick Start / 快速开始

Copy

```
1) Install dependencies (see above).
2) Export MONAI_LABEL_DATASTORE to your studies folder.
3) Run start_server.sh or your monailabel start command pointing to this app directory.
```

TCP Log Monitor / TCP 日志监控

Copy

```
from tcp_client import setup_logger
import atexit

# before starting the service
logger, tcp_listener = setup_logger("10.6.10.118", 2020, "LOG_DIRECTORY_PATH")

# ... start service ...

atexit.register(tcp_listener.stop)
```

MONAI Label App Architecture / 应用架构

1) Datastore Initialization / 数据仓库初始化

`self._datastore = self.init_datastore()` 创建 `datastore_v2.json` 追踪图像与标注；对象由 datastore 方法维护。可进一步通过缓存或动态标签更新进行优化。

Copy

```
{
  "name": "new-dataset",
  "description": "NEW Dataset",
  "images_dir": ".",
  "label_dir": "labels",
  "objects": {
    "IMAGE_FILE_NAME": {
      "image": { "ext": ".nii.gz", "info": { "ts": 1754010873, "name": "FILE_NAME_WITH_EXT" } },
      "labels": {}
    }
  }
}
```

2) Selection Strategies / 样本选择策略

`self._strategies = self.init_strategies()` 为每个未标注图像注册一个 `select_strategy`；前端通过 `Strategy.info()` 展示。

select_strategy

Copy

```
from monailabel.interfaces.task import Strategy

class select_strategy(Strategy):
    def __init__(self, description):
        super().__init__(description)

    def __call__(self, request, datastore):
        return {"id": self.description}
```

init_strategies()

Copy

```
import json, os

def init_strategies(self):
    with open(os.path.join(self.studies, "datastore_v2.json"), "r") as f:
        data = json.load(f)

    _strategies, i = {}, 0
    for img, vals in data.get("objects", {}).items():
        if vals.get("labels") == {}:
            _strategies[f"img_{i:04d}"] = select_strategy(img)
            i += 1
    return _strategies
```

next_sample()

Copy

```
def next_sample(self, request):
    # 1) Parse "img_xxxx" from request["strategy"]
    # 2) filename = self._strategies[img_name].info()["description"]
    # 3) path = self._datastore.get_image_uri(filename)
    # 4) return {"id": filename, "path": path}
    # 5) optionally mark the entry as selected (e.g., add a checkmark)
    return {"id": filename, "path": path}
```

3) Inference Registration / 推理注册

Copy

```
sam_infer = SAM2Infer(
    path=self.app_dir,
    network=None,
    type="deepedit",
    labels={"background": 0, "intersection": 1},
    dimension=3,
)
self.models = {"SAM": sam_infer}
return self.models
```

注意：Label 定义必须包含 `background: 0` 与 `intersection: 1`。

SAM2Infer (Core Inference Task) / 核心推理任务

Init / 初始化

从骨架 YAML 构建模型并加载权重。

__call__(request) Pipeline / 调用管线

1. Deep-copy request → data.
2. pre_transform(data): load prompts; minimal preprocessing for prompt parsing.
3. run_inferer(data): load image, read size (w,h,d), preprocess to `img_tensor`, build prompt configs, run inference, set `data['pred']`, `data['img_3d']`, `data['obj_ids']`.
4. post_transform(data): no-op (custom handling in writer).
5. writer(data): postprocess mask, save result, return `(output_path, result_json)`.

非法标签返回空白标签文件与占位 JSON。

Prompting & Inference Details / 提示与推理细节

Shape & Meta Tracking / 尺寸与元信息追踪

图像/掩膜/提示需统一缩放，并保留原始元信息以便在后处理阶段正确进行逆变换。

Multi-Point 3D Prompts / 多点 3D 提示

按 `z` 排序；相邻唯一样本 `z` 成对构造最大包围框；关键切片为 `(z_i + z_{i+1})/2`，并沿 +z/−z 广播；最小/最大 `z` 为严格起止边界，形成连续 3D 包围盒。

Forward/Backward Intersection / 前向/后向交集

同时执行前/后向推理，最终取其交集；对不确定体素保留一个额外标签。

Postprocessing / 后处理

执行最大连通域（LargGCC）；按标签值加权使背景为 0，在 3D Slicer 中不显示。

Front-end Integration (3D Slicer) / 前端集成

前端策略列表来自 `Strategy.info()`。点击"下一样本"会调用 `next_sample()`，返回所选文件的 `id` 与 `path`。

Extensibility & Optimization / 可扩展性与优化

- 主动学习：基于分数（Dice/熵/focal）的选择策略。
- 数据缓存：高频图像预加载；后台预取。
- 批处理与流式 IO：优化超大 3D 体数据读写。
- 结构化日志：通过 TCP 实时可视化。

Error Handling & Troubleshooting / 错误处理与故障排查

- Invalid labels → blank mask + dummy JSON.
- Strategy key mismatch → ensure `request['strategy']` matches registered keys.
- TorchIO compatibility → verify version (<0.3) if transforms fail.
- Path issues → use `datastore.get_image_uri()` for absolute path resolution.

- 非法标签 → 返回空掩膜与占位 JSON。
- 策略键不匹配 → 确保 `request['strategy']` 与注册键一致。
- TorchIO 兼容性 → 变换异常时核对版本（如需 <0.3）。
- 路径问题 → 使用 `datastore.get_image_uri()` 解析绝对路径。

Security & Privacy / 安全与隐私

涉及 PHI 需遵循机构安全策略。避免在日志中写入敏感元数据。通过 TCP 推送日志时，应限制仅授权客户端可见。

Known Limitations / 已知限制

- 假设已配置好 MedSAM2 权重与兼容的预处理流程。
- 代码块中的中文渲染受限（等宽字体），建议代码使用 ASCII。

Roadmap / 路线图

- 启动基于文件名选择的取样策略与前端交互。
- 用户从服务端获取图像以及进行3维点标注。
- 用户更新标注数据等待服务端推理。
- 服务端加载图片和标注，预处理图像以及提示参数列表。
- 服务端开始推理任务。
- 服务端将数据后处理后发往客户端。

License & Acknowledgements / 许可与致谢

算法基于 MedSAM2: : Segment Anything in 3D Medical Images and Videos 以及 monai label 开源平台。