一、项目简介与目标

目标：在**完全离线（内网/不可联网）**环境下，使用视觉大模型（CLIP）对监狱工具桌进行定期或按需检测，判断每个工具是否在其固定位置；若发现缺失则记录截图并触发告警或人工确认流程。

关键场景约束：

- 环境：不可联网（离线部署）、摄像头固定、光照可能有一定波动、不能频繁误报（误报代价高）。
- 实时性：**非实时 / 周期或按需检测**（可接受单次检测延迟秒级到几秒）。
- 可维护性：需提供配置界面调整 ROI、阈值与工具名词表。

二、总体架构（离线 CLIP 方案，主路线）

```
摄像头（定时/触发拍照）
        ↓（图片存储）
图像预处理模块（对齐、亮度校正）
        ↓
ROI 管理 + 图像裁剪（每个工具一个或多个 ROI）
        ↓
CLIP 推理模块（本地模型：图像编码器 + 文本编码器）
        ↓
相似度计算 → 阈值判定（每个 ROI 对应工具）
        ↓
规则引擎（多帧/多模板融合、异常抑制）
        ↓
事件记录（截图+相似度） → 报警 / 人工确认 / 日志
        ↓
管理界面/API（ROI 配置、阈值、历史记录、导出）
```

组件说明（交付要点）：

1. 摄像头：固定安装，保证同一视角；推荐 1080p，机械或软件支架锁定。
2. 预处理：图像对齐/仿射校正、白平衡/直方图均衡。
3. ROI 管理：每件工具在“基准图”上的一个或多个 ROI（可编辑保存）。
4. CLIP 引擎：本地已下载模型权重（ViT-B/32 推荐起步），文本模板库。
5. 规则引擎：对相似度做阈值判定、冗余（多模板、多帧）与误报抑制。
6. 管理/运维界面：查看历史、调整参数、导出 CSV/截图。
7. 安全：模型与日志存放在内网服务器，权限与审计。

三、模型与推理设计（详细）

3.1 选择与理由

- **模型**：OpenAI CLIP（建议 `ViT-B/32` 起步；若算力允许可用 ViT-L）。
  - 优点：无需类别标签的箱式重训练；能用文本模板灵活检验；离线可用。
- **为什么不直接用 YOLO**：YOLO 更适合实时定位并输出 bbox；但你不追求实时且希望“少训练/少标注”，CLIP 无需大量框标注更方便。若后续需要更高精度定位，可和 Grounding DINO / SAM 组合。

3.2 推理流程（每次检测）

1. 读取基准图与当前拍照图（或直接用当前图做推理）。
2. 预处理：对齐（若相机位置可能轻微抖动）、调整到统一分辨率（例如 640–768 最合适）。
3. 对每个 **ROI**（按配置）裁剪出子图（可以带一点 padding）。
4. 为每个工具准备若干文本模板（见下）并用 CLIP 文本编码器编码（可预先编码并缓存）：
   - 模板示例（中文/英文多语言都可）：
     - "a screwdriver on the table", "there is a screwdriver", "an empty slot where the screwdriver should be"（正/负模板）
     - 多模板能提高鲁棒性（短语、描述、材质、颜色）。
5. 对每个 ROI 的图像用 CLIP 图像编码器得到向量；计算与该工具所有文本模板的余弦相似度（或点积经 softmax）。
6. 聚合相似度：取文本模板的最大 similarity 作为 ROI→工具的得分；或对正/负模板分别求分并做差值（正分 - 负分）。
7. 判定：若得分 < 设定阈值（或正负差 < Δ），则判定“可能缺失”。
8. 冗余确认：若使用周期拍照，建议判定需连续 N 次（或多模板一致）才最终报警；也可要求人工复核一次（降低误报）。

3.3 文本模板策略（关键）

- 每个工具准备多条正描述和负描述（建议 3–6 条），示例：
  - 正：["a screwdriver on the table", "a flat-head screwdriver", "metal screwdriver lying horizontally"]

    - 负：["an empty slot", "no screwdriver", "an empty space"]
- 在模型输出上采用**最大正相似度**减去**最大负相似度**作为综合评分（更稳健）。

3.4 阈值建议（交付时需调优）

- 初始阈值示例（以 ViT-B/32、标准化相似度为依据）：
    - 正相似度阈值：0.25 ~ 0.35（若最大相似度 < threshold → 缺失）
    - 正负差阈值：0.1 ~ 0.2（若差值 < Δ → 无法判定/需要复核）
- **说明**：阈值需在验收集上调参（用样本集计算 PR 曲线并定目标误报率/漏报率）。

3.5 多模板、多帧融合（误报抑制）

- **多模板融合**：取 max 或 weighted average。
- **多帧融合**：例如要求 `k` 次连续检测均判定缺失（k≥2）或 `m` 次中超过阈值。
- **置信等级**：返回 HIGH/MEDIUM/LOW，便于决定是否自动报警或人工复核。

四、数据采集、验证与评估（交付给实现方的要求）

4.1 数据集

- **基准图集**：工具全部在位的清晰图（白天/夜间/灯光下各几个）。
- **负样本**：故意缺失某一工具的图（每个工具至少 50–100 张覆盖不同光照/角度）。
- **遮挡样本**：有人手伸入取放工具的帧。
- **环境变化样本**：有光照变化、桌面轻微杂物等。

4.2 标注与划分

- 对每张图记录：ToolName, ROI_ID, 是否在位（1/0），拍摄时间，光照标签。
- 划分：训练/验证/测试集（尽管 CLIP 不训练，仍用于阈值调优与评估）。

4.3 评价指标（建议）

- 精度（Precision）和召回（Recall）按“是否判定缺失”计算。
- F1、False Alarm Rate (FAR)、False Negative Rate (FNR)。

- 最重要：**误报率**（FAR）需非常低（例如 FAR < 1% 视场景而定）。
- AUC/PR 曲线用于阈值选择。

4.4 验收准则（示例，需与监狱方确认）

- 在测试集上，针对每个工具：Precision ≥ 0.95 或 Recall ≥ 0.95（按优先策略）。
- 总体误报率低于阈值（例如 <2%），并且人工复核流程可以在误报时快恢复。

五、离线部署细节（Mac 与 RK3588）

5.1 Mac（Intel / Apple Silicon）

- 推荐环境：Python 3.10+, PyTorch（支持 MPS），open_clip 或 transformers。
- 离线准备步骤：
  i. 在联网电脑上下载模型权重 `openai/clip-vit-base-patch32`（或 open_clip 的预训练文件）。
  ii. 将权重包与代码打包转移至内网 Mac（U盘或内网传输）。
  iii. 在 Mac 上安装依赖（若完全离线，需要预先打包 `.whl` 或使用本地 pip 缓存）。
- 运行方式：使用 `torch.device("mps")`（Apple Silicon）或 CPU。
- 性能预期：M1/M2/M3 上单张图推理在数百毫秒到 1s 范围（视分辨率与 batch）。

Mac 示例（推理模块伪代码）

（详见上面简短示例；交付可包含完整脚本与 CLI）

5.2 RK3588（嵌入式）

- 面临挑战：PyTorch 在 ARM 上跑 ViT 较慢；建议做模型量化与 NPU 加速。
- 推荐路径：
  i. 在联机机器把 CLIP 的**图像编码器**导出为 ONNX（只导图像部分）。
  ii. 使用 RKNN Toolkit 将 ONNX 转为 RKNN，量化为 INT8（使用 calibration dataset）。
  iii. 在板子上使用 RKNN runtime 调用 NPU 加速图像编码，文本编码器在 CPU 上做（文本模板少，CPU 开销较小）。
  iv. 若 RKNN 不可行，使用 `onnxruntime`（arm64）做 CPU 推理。
- 预处理、ROI 管理、规则引擎可在 CPU 上执行；NPU 只做 heavy 图像编码。

- 注意点：RKNN 转换需要匹配 ops 支持，可能对 ViT 仍需拆分或替换一部分算子；必要时可选更小的 CNN + CLIP-like 方法替代 ViT。

六、API / 管理界面 & 日志格式（交付接口规范）

6.1 REST API（示例）

- `POST /api/v1/detect`
  - 描述：上传图片或指令拍照，返回检测结果。
  - Request: `{ "image": base64|path, "mode":"single" }`
  - Response:


```
{
  "timestamp": "2025-09-27T00:00:00Z",
  "results": [
     {"roi_id":"R1","tool":"screwdriver","score":0.34,"status":"present","temp
     {"roi_id":"R2","tool":"wrench","score":0.12,"status":"missing"}
  ],
  "snapshots":["/data/snapshots/2025-09-27_0000_R2.jpg"]
}
```


- `GET /api/v1/rois`： 返回 ROI 列表与配置。
- `PUT /api/v1/rois/{id}`： 更新 ROI（坐标、工具名、templates）。
- `GET /api/v1/events`： 查询历史告警、导出 CSV。

6.2 日志格式（建议）

- CSV 或 JSONL，每条记录包含：
  - timestamp, camera_id, roi_id, tool_name, score, status, image_path, operator (if confirmed), note

七、安全、合规、离线交付要求

1. **离线模型转移**：所有模型与依赖在联网环境下载后，由可信人员通过物理介质（加密U盘）传入内网。保留传输记录。

2. **权限与认证**：管理界面与 API 只能内网访问，登录采用本地账号/密码并启用审计。
3. **数据保密**：图片和日志按监管要求存储（加密或受限访问）。
4. **防篡改**：设备（摄像头、主机）需物理上固定并标识，日志有完整时间戳与审计链。
5. **模型完整性**：对模型文件做哈希签名，部署时核对。

八、异常与回退策略（鲁棒性设计）

1. **误报降级**：若系统判定某工具缺失但人证/录像显示为手动操作，系统应允许管理员“一键异常恢复/忽略”并记录原因。
2. **光照异常检测**：若当前图整体亮度或差分图异常，触发“图像质量异常”而非直接报警。
3. **网络/设备断开**：摄像头或主机断开时写入本地故障日志并在恢复后同步。
4. **备份识别方案**：若 CLIP 异常，可切换到“模板匹配 + 差分检测”临时模式（代码内预留开关）。

九、交付物清单（交付给 Claude 实现时可直接据此开发）

（把以下逐项实现并交付代码/文档/工件）

1. 系统级设计文档（本文件 + 补充细节）。
2. 代码库（Python）模块：
    - 摄像头采集脚本（支持本地文件/USB/IP 摄像头）。
    - 预处理与对齐模块。
    - ROI 管理模块（JSON 格式 ROI 配置）。
    - CLIP 推理模块（支持 MPS/CPU，支持离线权重加载）。
    - 决策/规则引擎（多模板、多帧融合）。
    - REST API（Flask/FastAPI）与前端管理页面（简单 HTML/React）。
    - 单元测试 & 集成测试用例。
3. 模型权重及依赖包（需由你在联网环境下载后提供给实现者）。
4. 部署说明：Mac（MPS/CPU）与 RK3588（ONNX→RKNN）两条路详尽步骤。
5. 验收数据集与评估报告模板（PR 曲线、混淆矩阵）。
6. 运维手册（如何更新 ROI，如何更新模型，如何查看日志、如何处理误报）。

十、实施与交付建议（交接说明给 Claude）

把这份方案直接给 Claude 实现时，务必一并交付以下资料与权限：

- 摄像头样张（基准图、缺失样本、遮挡样本），用于阈值调优与本地测试。
- 内网环境的 Python 运行权限与目标机器（Mac / RK3588）的访问方式或镜像。
- 模型文件（CLIP 权重），以及任何需要离线安装的 Python wheel 包（若内网不能联网）。
- 明确验收指标（例如：不超过 X% 的误报率，或召回率达到 Y%）。
- 说明是否需要自动报警（即时）还是先由人工复核后报警（这会影响规则引擎的设计）。

十一、示例：核心示例代码（可直接交付实现者）

下面给出 **离线 CLIP 推理 + ROI 判定** 的可执行 Python 伪代码（用于 Mac/通用 CPU/MPS）——Claude 可据此实现生产代码并封装为 API。

```
# core_clip_inference.py （伪代码）
import torch
import open_clip
from PIL import Image
import numpy as np
import json
from pathlib import Path
from sklearn.preprocessing import normalize

# 加载模型（预下载好）
model_name = "ViT-B-32"
model, _, preprocess = open_clip.create_model_and_transforms(model_name, pretrained='o
tokenizer = open_clip.get_tokenizer(model_name)
device = "mps" if torch.backends.mps.is_available() else "cpu"
model.to(device).eval()

# 预加载文本模板（示例）
templates = {
  "screwdriver": {
     "pos": ["a screwdriver on the table", "a metal screwdriver"],
     "neg": ["no screwdriver", "an empty slot"]
  },
  "wrench": { "pos": ["a wrench on the table"], "neg": ["no wrench"] }
}

# 预编码文本模板
text_features = {}
with torch.no_grad():
    for tool, tdict in templates.items():
        pos_tokens = tokenizer(tdict["pos"]).to(device)
```

```
        neg_tokens = tokenizer(tdict["neg"]).to(device)
        pos_feat = model.encode_text(pos_tokens)  # (n_pos, d)
        neg_feat = model.encode_text(neg_tokens)
        text_features[tool] = {
           "pos": pos_feat / pos_feat.norm(dim=-1, keepdim=True),
           "neg": neg_feat / neg_feat.norm(dim=-1, keepdim=True)
        }

def infer_image_for_rois(image_path: str, rois: list, thresholds: dict):
    img = Image.open(image_path).convert("RGB")
    results = []
    with torch.no_grad():
        for roi in rois:
            # roi: {id, x,y,w,h, tool_name}
            crop = img.crop((roi['x'], roi['y'], roi['x']+roi['w'], roi['y']+roi['h']))
            inp = preprocess(crop).unsqueeze(0).to(device)  # (1,C,H,W)
            img_feat = model.encode_image(inp)  # (1,d)
            img_feat = img_feat / img_feat.norm(dim=-1, keepdim=True)  # normalize

            tool = roi['tool']
            pos_feats = text_features[tool]['pos']  # (n_pos, d)
            neg_feats = text_features[tool]['neg']  # (n_neg, d)

            # similarity
            sim_pos = (img_feat @ pos_feats.T).cpu().numpy().max()
            sim_neg = (img_feat @ neg_feats.T).cpu().numpy().max()
            score = float(sim_pos - sim_neg)

            status = "present" if sim_pos >= thresholds.get('pos', 0.25) and score >=
            results.append({
                "roi_id": roi['id'],
                "tool": tool,
                "sim_pos": float(sim_pos),
                "sim_neg": float(sim_neg),
                "score": score,
                "status": status
            })
    return results

# usage
rois = [
  {"id":"R1","x":10,"y":20,"w":200,"h":80,"tool":"screwdriver"},
  {"id":"R2","x":220,"y":20,"w":200,"h":80,"tool":"wrench"}
]
thresholds = {"pos":0.28, "diff":0.12}
print(infer_image_for_rois("capture.jpg", rois, thresholds))
```

> 注：上面代码在实际实现时需要加入：图像对齐、异常检测、日志、错误处理、并发/批量推理、缓存文本特征、以及 API 封装等。

十二、交付与验收（交接 Claude 时的验收清单）

- 可运行的代码仓库（含安装、运行说明）。
- Mac 与 RK3588 的部署脚本与步骤说明（包括 ONNX→RKNN 详细脚本或说明）。
- 测试数据与评估报告（PR 曲线、混淆矩阵、样本截图）。
- 管理界面（或最小可用 CLI）能：配置 ROI、调整阈值、查看历史事件与截图。
- 运维手册与安全说明。