1) 创建工程目录与项目骨架

建议把这个项目单独放一个子目录，结构更清晰：

```
mkdir -p ~/Project/PythonProject/rpi_vision_gradio
cd ~/Project/PythonProject/rpi_vision_gradio
```

```
pi@raspberrypi:~ $ mkdir -p ~/Project/PythonProject/rpi_vision_gradio
cd ~/Project/PythonProject/rpi_vision_gradio
```

UV安装
在bash菜单使用

更新树莓派

```
sudo apt update && sudo apt upgrade
```

使用中转一键安装脚本

```
curl -LsSf https://gitee.com/wangnov/uv-custom/releases/download/0.9.18/uv-installer-custom.sh | sh
```

初始化一个 uv 项目（会生成 `pyproject.toml` 等）：

```
uv init
```

```
pi@raspberrypi:~/Project/PythonProject/rpi_vision_gradio $ uv init
Initialized project `rpi-vision-gradio`
```

创建虚拟环境（推荐放在项目内 `.venv`）：

```
uv venv
```

```
pi@raspberrypi:~/Project/PythonProject/rpi_vision_gradio $ uv venv
Using CPython 3.11.2 interpreter at: /usr/bin/python3.11
Creating virtual environment at: .venv
Activate with: source .venv/bin/activate
```

激活虚拟环境（.venv`）：

```
source .venv/bin/activate
```

```
pi@raspberrypi:~/Project/PythonProject/rpi_vision_gradio $ source .venv/bin/activate
(rpi_vision_gradio) pi@raspberrypi:~/Project/PythonProject/rpi_vision_gradio $
```

2) 系统级依赖安装（树莓派上很关键）

2.1 摄像头相关工具（建议装）

```
sudo apt update
sudo apt install -y v4l-utils
```

```
(rpi_vision_gradio) pi@raspberrypi:~/Project/PythonProject/rpi_vision_gradio $ sudo apt install -y v4l-utils
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
v4l-utils is already the newest version (1.22.1-5+b2).
0 upgraded, 0 newly installed, 0 to remove and 162 not upgraded.
```

你可以用它确认有没有 `/dev/video0`：

```
v4l2-ctl --list-devices
ls -l /dev/video*
```

```
USB 2.0 Camera: HD USB Camera (usb-0000:01:00.0-1.1):
        /dev/video0
        /dev/video1
        /dev/media4
```

3) 用 uv 安装 Python 依赖（推荐 headless）

在项目目录：

```
cd ~/Project/PythonProject/rpi_vision_gradio
```

安装依赖（推荐）：

```
uv pip install "gradio>=4" numpy opencv-python
```

验证一下：

```
uv run python -c "import cv2,gradio,numpy; print('cv2',cv2.__version__)"
```

```
crw-rw----+ 1 root video 81, 13 Dec 27 22:17 /dev/video31
(PythonProject) pi@raspberrypi:~/Project/CProject/rpi_vision_cpp/build $ uv run python -c "import cv2,gradio,numpy; print('cv
2',cv2.__version__)"

cv2 4.12.0
```

4) 放置你的程序代码（app.py）

把你上传的这一体化脚本保存成项目里的 `app.py`（内容就是你当前这份 Gradio + OpenCV 端侧处理脚本）

在目录下创建文件：

```
nano app.py
```

把脚本粘进去保存退出。

```
import cv2
import gradio as gr
import numpy as np
import threading
import time
import json

# ======================
# 工具函数：颜色/HSV/主色
# ======================
def hex_to_rgb(hex_color: str):
    hex_color = hex_color.lstrip("#")
    r = int(hex_color[0:2], 16)
    g = int(hex_color[2:4], 16)
    b = int(hex_color[4:6], 16)
    return (r, g, b)

def rgb_to_hex(rgb):
    r, g, b = [int(x) for x in rgb]
    return "#{:02X}{:02X}{:02X}".format(r, g, b)

def rgb_to_hsv_opencv(rgb):
    r, g, b = rgb
    bgr = np.uint8([[[b, g, r]]])
    hsv = cv2.cvtColor(bgr, cv2.COLOR_BGR2HSV)[0, 0]
    return (int(hsv[0]), int(hsv[1]), int(hsv[2]))  # H:0-179

def build_hsv_ranges(center_hsv, h_tol, s_tol, v_tol):
    """
    OpenCV HSV: H 0-179, S 0-255, V 0-255
    处理 Hue 环绕（红色跨 0/179）
    """
    h, s, v = center_hsv
    h_tol = int(h_tol); s_tol = int(s_tol); v_tol = int(v_tol)

    s1 = max(0, s - s_tol); s2 = min(255, s + s_tol)
    v1 = max(0, v - v_tol); v2 = min(255, v + v_tol)

    h1 = h - h_tol
    h2 = h + h_tol

    ranges = []
    if h1 < 0:
        ranges.append({"lower": [0, s1, v1], "upper": [min(179, h2), s2, v2]})
        ranges.append({"lower": [180 + h1, s1, v1], "upper": [179, s2, v2]})
    elif h2 > 179:
        ranges.append({"lower": [max(0, h1), s1, v1], "upper": [179, s2, v2]})
        ranges.append({"lower": [0, s1, v1], "upper": [h2 - 180, s2, v2]})
```

```
    else:
        ranges.append({"lower": [h1, s1, v1], "upper": [h2, s2, v2]})
    return ranges

def build_mask_from_hsv_ranges(bgr, hsv_ranges):
    hsv = cv2.cvtColor(bgr, cv2.COLOR_BGR2HSV)
    mask = None
    for r in hsv_ranges:
        lower = np.array(r["lower"], dtype=np.uint8)
        upper = np.array(r["upper"], dtype=np.uint8)
        m = cv2.inRange(hsv, lower, upper)
        mask = m if mask is None else cv2.bitwise_or(mask, m)
    mask = cv2.medianBlur(mask, 5)
    return mask

def dominant_color_kmeans(bgr_roi, k=3, sample_max=4000, downscale=(80, 80)):
    """
    ROI 主色（KMeans，做下采样+随机采样降低开销）
    返回 RGB (0-255)
    """
    if bgr_roi is None or bgr_roi.size == 0:
        return None

    # 下采样降低计算量
    try:
        roi_small = cv2.resize(bgr_roi, downscale, interpolation=cv2.INTER_AREA)
    except:
        roi_small = bgr_roi

    rgb = cv2.cvtColor(roi_small, cv2.COLOR_BGR2RGB)
    pixels = rgb.reshape(-1, 3)

    n = pixels.shape[0]
    if n > sample_max:
        idx = np.random.choice(n, sample_max, replace=False)
        pixels = pixels[idx]

    pixels = np.float32(pixels)

    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 20, 1.0)
    _, labels, centers = cv2.kmeans(pixels, int(k), None, criteria, 2,
cv2.KMEANS_PP_CENTERS)

    labels = labels.flatten()
    counts = np.bincount(labels, minlength=int(k))
    dom = centers[np.argmax(counts)]
    dom = np.clip(dom, 0, 255).astype(np.uint8)
    return (int(dom[0]), int(dom[1]), int(dom[2]))

def swatch_html(hex_color):
    if not hex_color:
        return "<div style='height:42px;border:1px solid #ccc;border-radius:10px;'></div>"
    return f"""
```

```
    <div style="height:42px;border-radius:10px;border:1px solid #ccc;background:{hex_color};
                display:flex;align-items:center;justify-content:center;font-weight:600;">
        {hex_color}
    </div>
    """

def clamp(v, lo, hi):
    return max(lo, min(hi, v))


# ======================
# 全局状态
# ======================
class GlobalState:
    def __init__(self):
        self.lock = threading.Lock()
        self.running = True

        self.latest_raw_bgr = None
        self.latest_raw_rgb = None

        self.latest_out_rgb = None
        self.status = "等待相机..."
        self.meta = {}
        self.dom_hex = ""
        self.dom_rgb = None

        self.click_action = "None"   # None / PickColor / SetROI
        self.roi_points = []

        self.frame_count = 0
        self.last_dom_calc_frame = -999999

        self.cfg = self.default_cfg()

    @staticmethod
    def default_cfg():
        return {
            # ROI / 主色
            "roi": None,                # [x1,y1,x2,y2]
            "draw_roi": True,
            "compute_dominant": True,
            "dominant_k": 3,
            "dominant_every_n_frames": 6,  # 每 N 帧计算一次主色，省 CPU

            # 颜色报警
            "enable_color_alarm": False,
            "hsv_ranges": [],
            "alarm_pixel_threshold": 3000,
            "show_mask_preview": False,

            # 图像算法
            "enable_blur": False,
```

```
            "blur_ksize": 5,

            "enable_edges": False,
            "canny_t1": 80,
            "canny_t2": 160,
            "edges_mode": "overlay",     # overlay / only
            "edges_alpha": 0.6,
        }

state = GlobalState()


# ======================
# 核心处理：都在树莓派本机
# ======================
def apply_processing(frame_bgr, cfg, frame_count):
    h, w = frame_bgr.shape[:2]
    out = frame_bgr.copy()
    meta = {}

    # ROI
    roi = cfg.get("roi")
    roi_bgr = frame_bgr
    if roi and len(roi) == 4:
        x1, y1, x2, y2 = roi
        x1 = clamp(int(x1), 0, w - 1)
        x2 = clamp(int(x2), 0, w)
        y1 = clamp(int(y1), 0, h - 1)
        y2 = clamp(int(y2), 0, h)
        if x2 > x1 and y2 > y1:
            roi_bgr = frame_bgr[y1:y2, x1:x2]
            meta["roi"] = [x1, y1, x2, y2]
            if cfg.get("draw_roi", True):
                cv2.rectangle(out, (x1, y1), (x2, y2), (255, 0, 0), 2)

    # 颜色报警（在原图上做更稳）
    alarm = False
    alarm_pixels = 0
    if cfg.get("enable_color_alarm", False) and cfg.get("hsv_ranges"):
        mask = build_mask_from_hsv_ranges(frame_bgr, cfg["hsv_ranges"])
        alarm_pixels = int(cv2.countNonZero(mask))
        alarm = alarm_pixels > int(cfg.get("alarm_pixel_threshold", 3000))
        meta["alarm"] = alarm
        meta["alarm_pixels"] = alarm_pixels

        if cfg.get("show_mask_preview", False):
            out = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

    # 模糊
    if cfg.get("enable_blur", False):
        k = int(cfg.get("blur_ksize", 5))
        if k % 2 == 0:
            k += 1
```

```
        out = cv2.GaussianBlur(out, (k, k), 0)

    # 边缘检测（在 out 上做）
    if cfg.get("enable_edges", False):
        t1 = int(cfg.get("canny_t1", 80))
        t2 = int(cfg.get("canny_t2", 160))
        gray = cv2.cvtColor(out, cv2.COLOR_BGR2GRAY)
        edges = cv2.Canny(gray, t1, t2)
        edges_bgr = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

        if cfg.get("edges_mode", "overlay") == "only":
            out = edges_bgr
        else:
            alpha = float(cfg.get("edges_alpha", 0.6))
            out = cv2.addWeighted(out, 1.0 - alpha, edges_bgr, alpha, 0)

    return out, meta, roi_bgr, alarm, alarm_pixels


# ======================
# 相机线程：采集 + 处理（全部在树莓派）
# ======================
def camera_loop(camera_id=0, width=640, height=480, target_fps=20):
    cap = cv2.VideoCapture(camera_id)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)

    if not cap.isOpened():
        with state.lock:
            state.status = "❌ 相机打开失败：请检查 /dev/video0 或相机配置"
        return

    frame_interval = 1.0 / max(1, target_fps)

    while state.running:
        t0 = time.time()
        ret, frame = cap.read()
        if not ret:
            time.sleep(0.05)
            continue

        with state.lock:
            cfg = dict(state.cfg)
            state.frame_count += 1
            fc = state.frame_count

        # 处理
        out_bgr, meta, roi_bgr, alarm, alarm_pixels = apply_processing(frame, cfg, fc)

        # 主色（按间隔算一次）
        dom_hex = ""
        dom_rgb = None
        if cfg.get("compute_dominant", True):
```

```
            every = int(cfg.get("dominant_every_n_frames", 6))
            # 每 every 帧算一次
            if fc % max(1, every) == 0:
                dom_rgb = dominant_color_kmeans(
                    roi_bgr,
                    k=int(cfg.get("dominant_k", 3)),
                    sample_max=4000,
                    downscale=(80, 80)
                )
                if dom_rgb:
                    dom_hex = rgb_to_hex(dom_rgb)
            else:
                # 不计算时沿用上一帧结果
                with state.lock:
                    dom_hex = state.dom_hex
                    dom_rgb = state.dom_rgb
        else:
            dom_hex, dom_rgb = "", None

        # 叠字（可选：让画面更直观）
        if dom_hex:
            cv2.putText(out_bgr, f"DOM {dom_hex}", (10, 28),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 255), 2)
        if alarm:
            cv2.putText(out_bgr, f"ALARM pixels={alarm_pixels}", (10, 56),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2)

        # 更新全局显示数据
        out_rgb = cv2.cvtColor(out_bgr, cv2.COLOR_BGR2RGB)
        raw_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        with state.lock:
            state.latest_raw_bgr = frame
            state.latest_raw_rgb = raw_rgb
            state.latest_out_rgb = out_rgb

            state.meta = meta
            state.dom_hex = dom_hex
            state.dom_rgb = dom_rgb

            if alarm:
                state.status = f"🚨 报警触发 pixels={alarm_pixels}"
            else:
                state.status = "✅ 正常"

        # 控制帧率
        dt = time.time() - t0
        sleep_t = frame_interval - dt
        if sleep_t > 0:
            time.sleep(sleep_t)

    cap.release()
```

```
threading.Thread(target=camera_loop, daemon=True).start()


# ======================
# Gradio：流式刷新
# ======================
def stream_ui():
    while True:
        time.sleep(0.05)
        with state.lock:
            frame = state.latest_out_rgb
            status = state.status
            dom_hex = state.dom_hex
            roi = state.cfg.get("roi")
        roi_text = f"{roi}" if roi else "未设置"
        yield frame, status, (dom_hex or ""), swatch_html(dom_hex), roi_text

def stream_debug():
    while True:
        time.sleep(0.2)
        with state.lock:
            cfg = dict(state.cfg)
            meta = dict(state.meta)
        yield json.dumps(cfg, ensure_ascii=False, indent=2), json.dumps(meta,
ensure_ascii=False, indent=2)


# ======================
# UI 交互：点击动作 & 画面点击
# ======================
def set_click_action(action):
    with state.lock:
        state.click_action = action
        state.roi_points = []
    if action == "PickColor":
        return "已开启：点击取色（去上方画面点一下）"
    if action == "SetROI":
        return "已开启：点击框选ROI（点两下确定矩形对角）"
    return "点击动作已关闭（None）"

def on_image_click(evt: gr.SelectData, h_tol, s_tol, v_tol):
    x, y = evt.index

    with state.lock:
        action = state.click_action
        raw = state.latest_raw_rgb.copy() if state.latest_raw_rgb is not None else None

    if raw is None:
        return "还没有画面，无法取色/框选。"

    h, w = raw.shape[:2]
    x = clamp(int(x), 0, w - 1)
```

```
    y = clamp(int(y), 0, h - 1)

    if action == "PickColor":
        rgb = raw[y, x].tolist()  # [R,G,B]
        hsv = rgb_to_hsv_opencv(tuple(rgb))
        ranges = build_hsv_ranges(hsv, h_tol, s_tol, v_tol)
        with state.lock:
            state.cfg["hsv_ranges"] = ranges
            state.cfg["enable_color_alarm"] = True
        return f"✅ 取色成功：RGB={rgb} HSV中心={hsv}（已启用颜色报警）"

    if action == "SetROI":
        with state.lock:
            state.roi_points.append((x, y))
            pts = list(state.roi_points)

        if len(pts) < 2:
            return f"ROI 第1点：({pts[0][0]}, {pts[0][1]})，再点一次确定对角"

        (x1, y1), (x2, y2) = pts[:2]
        x1, x2 = sorted([x1, x2])
        y1, y2 = sorted([y1, y2])

        with state.lock:
            state.cfg["roi"] = [x1, y1, x2, y2]
            state.roi_points = []

        return f"✅ ROI 已设置：[{x1},{y1},{x2},{y2}]"

    return "当前点击动作=None（到对应 Tab 开启：取色 或 框选ROI）"


# ======================
# UI：颜色报警/ROI/算法/高级
# ======================
def apply_preset_color(name):
    presets = {
        "Red": [
            {"lower": [0, 120, 70], "upper": [10, 255, 255]},
            {"lower": [170, 120, 70], "upper": [179, 255, 255]},
        ],
        "Yellow": [{"lower": [20, 100, 100], "upper": [30, 255, 255]}],
        "Green": [{"lower": [35, 100, 100], "upper": [85, 255, 255]}],
        "None": []
    }
    with state.lock:
        state.cfg["hsv_ranges"] = presets.get(name, [])
        state.cfg["enable_color_alarm"] = (name != "None")
    return f"✅ 已设置预置颜色：{name}"

def apply_palette_color(hex_color, h_tol, s_tol, v_tol):
    rgb = hex_to_rgb(hex_color)
    hsv = rgb_to_hsv_opencv(rgb)
```

```
    ranges = build_hsv_ranges(hsv, h_tol, s_tol, v_tol)
    with state.lock:
        state.cfg["hsv_ranges"] = ranges
        state.cfg["enable_color_alarm"] = True
    return f"✅ 调色板 {hex_color} -> HSV中心={hsv}（已启用颜色报警）"

def clear_alarm_target():
    with state.lock:
        state.cfg["hsv_ranges"] = []
        state.cfg["enable_color_alarm"] = False
    return "✅ 已清除报警目标（关闭颜色报警）"

def clear_roi():
    with state.lock:
        state.cfg["roi"] = None
        state.roi_points = []
    return "✅ ROI 已清除"

def use_dominant_as_target(h_tol, s_tol, v_tol):
    with state.lock:
        dom_rgb = state.dom_rgb
    if not dom_rgb:
        return "还没有主色结果。请先框选 ROI，并等待 dominant_hex 显示。"
    hsv = rgb_to_hsv_opencv(tuple(dom_rgb))
    ranges = build_hsv_ranges(hsv, h_tol, s_tol, v_tol)
    with state.lock:
        state.cfg["hsv_ranges"] = ranges
        state.cfg["enable_color_alarm"] = True
    return f"✅ 已将 ROI 主色 RGB={dom_rgb} (HSV中心={hsv}) 设为报警目标"

def apply_algo_settings(
    enable_alarm, alarm_pixel_threshold, show_mask_preview,
    enable_edges, edges_mode, canny_t1, canny_t2, edges_alpha,
    enable_blur, blur_ksize,
    draw_roi, compute_dominant, dominant_k, dominant_every
):
    with state.lock:
        state.cfg["enable_color_alarm"] = bool(enable_alarm)
        state.cfg["alarm_pixel_threshold"] = int(alarm_pixel_threshold)
        state.cfg["show_mask_preview"] = bool(show_mask_preview)

        state.cfg["enable_edges"] = bool(enable_edges)
        state.cfg["edges_mode"] = edges_mode
        state.cfg["canny_t1"] = int(canny_t1)
        state.cfg["canny_t2"] = int(canny_t2)
        state.cfg["edges_alpha"] = float(edges_alpha)

        state.cfg["enable_blur"] = bool(enable_blur)
        state.cfg["blur_ksize"] = int(blur_ksize)

        state.cfg["draw_roi"] = bool(draw_roi)
        state.cfg["compute_dominant"] = bool(compute_dominant)
        state.cfg["dominant_k"] = int(dominant_k)
```

```
            state.cfg["dominant_every_n_frames"] = int(dominant_every)

        return "✅ 算法配置已应用（全部在树莓派端执行）"

def reset_all_config():
    with state.lock:
        state.cfg = state.default_cfg()
        state.roi_points = []
        state.click_action = "None"
    return "✅ 已恢复默认配置（并关闭点击动作）"


# =======================
# Gradio 布局：分 Tab
# =======================
css = """
#topcard .gr-box {border-radius: 16px;}
"""

with gr.Blocks(title="树莓派一体化监控系统（端侧计算）", css=css) as demo:
    gr.Markdown("# 🤖 树莓派一体化监控系统（采集 + 计算 + 控制界面都在树莓派）")

    # 顶部固定区：画面 + 关键状态
    with gr.Row(elem_id="topcard"):
        with gr.Column(scale=3):
            video = gr.Image(label="实时画面（树莓派本机处理输出）", type="numpy", height=480)
        with gr.Column(scale=2):
            status = gr.Textbox(label="系统状态", value="初始化中...", lines=2)

            with gr.Row():
                dom_hex = gr.Textbox(label="ROI 主色 dominant_hex", value="", scale=2)
                dom_swatch = gr.HTML(value=swatch_html(None), scale=1)

            roi_box = gr.Textbox(label="当前 ROI", value="未设置", lines=1)
            click_hint = gr.Label(label="点击反馈", value="到下方 Tab 开启：取色 或 框选ROI")

    gr.Markdown("---")

    with gr.Tabs():
        with gr.Tab("📺 监控"):
            gr.Markdown(
                "- 上方实时画面为最终输出（在树莓派上完成算法）\n"
                "- 颜色报警：到 **🎯 颜色报警**\n"
                "- 框选 ROI/主色：到 **📦 ROI / 主色**\n"
                "- 边缘/模糊等：到 **🧹 图像算法**\n"
            )

        with gr.Tab("🎯 颜色报警"):
            with gr.Row():
                with gr.Column(scale=2):
                    preset = gr.Radio(
                        choices=["Red", "Yellow", "Green", "None"],
                        value="None",
```

```
                        label="预置颜色"
                    )
                    preset_info = gr.Label(label="预置反馈")

                    palette = gr.ColorPicker(label="调色板选色", value="#FF0000")
                    h_tol = gr.Slider(0, 60, value=10, step=1, label="Hue 容差")
                    s_tol = gr.Slider(0, 255, value=80, step=1, label="S 容差")
                    v_tol = gr.Slider(0, 255, value=80, step=1, label="V 容差")

                    with gr.Row():
                        palette_apply = gr.Button("应用调色板为报警目标", variant="primary")
                        clear_alarm_btn = gr.Button("清除报警目标")
                    palette_info = gr.Label(label="调色板反馈")

                with gr.Column(scale=1):
                    gr.Markdown("### 点击取色")
                    pick_action_btn = gr.Button("开启：点击画面取色", variant="primary")
                    pick_stop_btn = gr.Button("关闭点击动作")

                    gr.Markdown(
                        "提示：开启后，到上方画面点一下取样。\n"
                        "会用 HSV 容差生成阈值并启用报警。"
                    )

        with gr.Tab("🧊 ROI / 主色"):
            with gr.Row():
                with gr.Column(scale=1):
                    roi_action_btn = gr.Button("开启：点击画面框选 ROI", variant="primary")
                    roi_clear_btn = gr.Button("清除 ROI")
                    roi_stop_btn = gr.Button("关闭点击动作")

                    use_dom_btn = gr.Button("使用当前 ROI 主色作为报警目标")
                    dom_info = gr.Label(label="主色→报警反馈")

                with gr.Column(scale=1):
                    gr.Markdown(
                        "操作：\n"
                        "1) 开启框选 ROI → 上方画面点两下\n"
                        "2) 等 dominant_hex 显示\n"
                        "3) 点“主色作为报警目标”即可\n"
                    )

        with gr.Tab("🍼 图像算法"):
            gr.Markdown("所有算法均在树莓派端执行（节省网络/延迟）")

            with gr.Accordion("颜色报警（算法层）", open=True):
                enable_alarm = gr.Checkbox(value=False, label="启用颜色报警（需已设置
hsv_ranges）")
                alarm_pixel_threshold = gr.Slider(0, 50000, value=3000, step=50, label="报警
像素阈值")
                show_mask_preview = gr.Checkbox(value=False, label="显示 mask 预览（替换输出画
面）")
```

```
            with gr.Accordion("边缘检测（Canny）", open=False):
                enable_edges = gr.Checkbox(value=False, label="启用边缘检测")
                edges_mode = gr.Radio(choices=["overlay", "only"], value="overlay",
label="显示方式")
                canny_t1 = gr.Slider(0, 500, value=80, step=1, label="阈值1")
                canny_t2 = gr.Slider(0, 500, value=160, step=1, label="阈值2")
                edges_alpha = gr.Slider(0.0, 1.0, value=0.6, step=0.05, label="Overlay 叠加强
度")

            with gr.Accordion("模糊", open=False):
                enable_blur = gr.Checkbox(value=False, label="启用高斯模糊")
                blur_ksize = gr.Slider(1, 31, value=5, step=2, label="核大小(奇数)")

            with gr.Accordion("ROI/主色", open=False):
                draw_roi = gr.Checkbox(value=True, label="绘制 ROI 框")
                compute_dominant = gr.Checkbox(value=True, label="计算 ROI 主色")
                dominant_k = gr.Slider(1, 8, value=3, step=1, label="主色聚类 K")
                dominant_every = gr.Slider(1, 30, value=6, step=1, label="每 N 帧计算主色（省
CPU）")

            apply_algo_btn = gr.Button("应用算法配置", variant="primary")
            algo_info = gr.Label(label="算法反馈")

        with gr.Tab("⚙️ 高级/调试"):
            reset_btn = gr.Button("重置为默认配置", variant="stop")
            reset_info = gr.Label(label="重置反馈")
            with gr.Row():
                cfg_json = gr.Textbox(label="当前配置（JSON）", lines=18)
                meta_json = gr.Textbox(label="实时 meta（JSON）", lines=18)

    # ========== 事件绑定 ==========
    video.select(fn=on_image_click, inputs=[h_tol, s_tol, v_tol], outputs=click_hint)

    pick_action_btn.click(fn=set_click_action, inputs=gr.State("PickColor"),
outputs=click_hint)
    roi_action_btn.click(fn=set_click_action, inputs=gr.State("SetROI"), outputs=click_hint)
    pick_stop_btn.click(fn=set_click_action, inputs=gr.State("None"), outputs=click_hint)
    roi_stop_btn.click(fn=set_click_action, inputs=gr.State("None"), outputs=click_hint)

    preset.change(fn=apply_preset_color, inputs=preset, outputs=preset_info)
    palette_apply.click(fn=apply_palette_color, inputs=[palette, h_tol, s_tol, v_tol],
outputs=palette_info)
    clear_alarm_btn.click(fn=clear_alarm_target, inputs=None, outputs=palette_info)

    roi_clear_btn.click(fn=clear_roi, inputs=None, outputs=click_hint)
    use_dom_btn.click(fn=use_dominant_as_target, inputs=[h_tol, s_tol, v_tol],
outputs=dom_info)

    apply_algo_btn.click(
        fn=apply_algo_settings,
        inputs=[
            enable_alarm, alarm_pixel_threshold, show_mask_preview,
            enable_edges, edges_mode, canny_t1, canny_t2, edges_alpha,
```

```
            enable_blur, blur_ksize,
            draw_roi, compute_dominant, dominant_k, dominant_every
        ],
        outputs=algo_info
    )

    reset_btn.click(fn=reset_all_config, inputs=None, outputs=reset_info)

    # 自动刷新
    demo.load(stream_ui, None, outputs=[video, status, dom_hex, dom_swatch, roi_box])
    demo.load(stream_debug, None, outputs=[cfg_json, meta_json])


if __name__ == "__main__":
    # 允许局域网访问：手机/PC 浏览器打开 http://树莓派IP:7860
    demo.launch(server_navigation="0.0.0.0", server_port=7860)
```

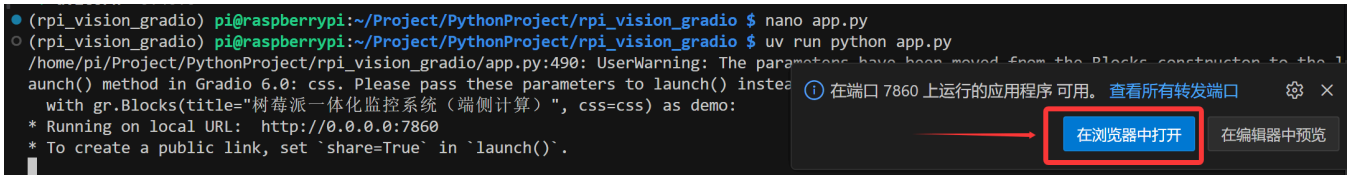
5) 运行（本机/局域网访问）

在项目目录运行：

```
uv run python app.py
```

然后在浏览器打开：

- 本机：`http://127.0.0.1:7860`
- 局域网其他设备：`http://<树莓派IP>:7860`



查看树莓派 IP：

```
hostname -I
```

6) 使用说明

- **点击取色**：到“🎯 颜色报警”Tab → 点“开启：点击画面取色”→ 去上方图像点一下
- **框选 ROI**：到“🧊 ROI / 主色”Tab → 点“开启：点击画面框选 ROI”→ 去上方图像点两下（对角）
- **主色显示**：ROI 设置后会显示 dominant_hex（并在画面叠字）
- **边缘检测/模糊**：到“🪄 图像算法”Tab 勾选参数并应用