```
# --- 推論 ---
def load_interpreter(path):
    global INTERPRETER, IN_D, OUT_D
    INTERPRETER = tflite.Interpreter(model_path=path)
    INTERPRETER.allocate_tensors()
    IN_D = INTERPRETER.get_input_details()[0]
    OUT_D = INTERPRETER.get_output_details()[0]
    log.info(f"Loaded TFLite model: {path}")
def load_scaler():
    global SCALER_MEAN, SCALER_STD
    SCALER_MEAN = np.load(SCALER_MEAN_PATH)
    SCALER_STD = np.load(SCALER_STD_PATH)
    log.info("Scaler loaded.")
def run_inference(X):
    global LAST_STATE
    if DATATYPE == "int8":
        scale, zp = IN_D["quantization"]
        xq = np.round(X / scale + zp).astype(np.int8)
        INTERPRETER.set_tensor(IN_D["index"], xq)
    else:
        INTERPRETER.set_tensor(IN_D["index"], X.astype(np.float32))
    t0 = time.perf_counter()
    INTERPRETER.invoke()
    dt = time.perf_counter() - t0
    if DATATYPE == "int8":
        out_scale, out_zp = OUT_D["quantization"]
        recon = INTERPRETER.get_tensor(OUT_D["index"]).astype(np.float32)
        recon = (recon - out_zp) * out_scale
        x_rec = (xq.astype(np.float32) - zp) * scale
        mse = float(np.mean((x_rec - recon) ** 2))
    else:
        recon = INTERPRETER.get_tensor(OUT_D["index"]).astype(np.float32)
        mse = float(np.mean((X - recon) ** 2))
    fps = 1.0 / dt if dt > 0 else 0
    payload = {
        "ts": int(time.time() * 1e9),
        "mse": mse,
        "thr": THRESHOLD,
        "win": WINDOW_LEN,
        "fps": fps,
    }
    MQTT.publish(TELEM_TOPIC, json.dumps(payload), qos=0)
    state = "ANOMALY" if mse > THRESHOLD else "NORMAL"
    if state == "ANOMALY" and LAST_STATE == "NORMAL":
        ev = {
            "ts": payload["ts"],
            "mse": mse,
            "thr": THRESHOLD,
            "win": WINDOW_LEN,
            "type": "anomaly",
        }
        MQTT.publish(EVENT_TOPIC, json.dumps(ev), qos=1)
    LAST_STATE = state
def publish_status(retain=False):
    payload = {
        "device_id": DEVICE_ID,
        "state": "OK",
        "datatype": DATATYPE,
        "threshold": THRESHOLD,
        "k_clip": K_CLIP,
        "win": WINDOW_LEN,
        "ts": int(time.time() * 1e9),
    }
    MQTT.publish(STATUS_TOPIC, json.dumps(payload), qos=1, retain=retain)
:
```