```
: (省略)
class RawWindowPublisher:
    def __init__(self, cfg_path: str):
        self.cfg = load_yaml(cfg_path)
        self.device_id = str(self.cfg.get("device_id", "pi5-001"))
        m = self.cfg.get("mqtt", {}) or {}
        self.base_topic = m.get("base_topic", "anomdet/v1/siteA/line1/machineX")

        # Topics
        self.cmd_topic    = f"{self.base_topic}/{self.device_id}/cmd"
        self.ack_topic    = f"{self.base_topic}/{self.device_id}/ack"
        self.status_topic = f"{self.base_topic}/{self.device_id}/status"
        self.win_topic    = f"{self.base_topic}/{self.device_id}/window/raw"

        # Window length
        self.window_len = int(self.cfg.get("window_len", 10))

        # State
        self.running = True

        # Hardware
        self.sensor = ADXL345()

        # MQTT
        self.mqtt = mqtt.Client(callback_api_version=mqtt.CallbackAPIVersion.VERSION2,
client_id=f"edge-rawpub-{self.device_id}")
    if m.get("username") and m.get("password"):
        self.mqtt.username_pw_set(m["username"], m["password"])
    self.mqtt.on_message = self._on_message
    self.mqtt.on_connect = self._on_connect
    self.mqtt.connect(m.get("host","127.0.0.1"), int(m.get("port",1883)), keepalive=30)
    self.mqtt.loop_start()

# ---------- MQTT ----------
def _on_connect(self, client, userdata, flags, rc, properties=None):
    client.subscribe(self.cmd_topic, qos=1)
    self._publish_status(retain=True, state="OK")

def _on_message(self, client, userdata, msg):
    # Edge currently doesn't implement commands (reserved for future use)
    pass

def _publish_status(self, retain: bool=False, state: str="OK"):
    self.mqtt.publish(self.status_topic, json.dumps({
        "device_id": self.device_id,
        "state": state,
        "win": self.window_len,
        "ts": now_ns()
    }), qos=1, retain=retain)

# ---------- Main ----------
def start(self):
    period = 0.1
    next_time = time.time()
    queue = deque(maxlen=self.window_len)

    try:
        while self.running:
            # Read sensor
            x, y, z = self.sensor.read_axes()
            queue.append([float(x), float(y), float(z)])

            # If window ready → publish RAW
            if len(queue) == self.window_len:
                payload = {"ts": now_ns(), "raw": list(queue), "win": self.window_len}
                self.mqtt.publish(self.win_topic, json.dumps(payload), qos=0)

            # Deterministic pacing
            next_time += period
            sleep = next_time - time.time()
            if sleep > 0:
                time.sleep(sleep)
            else:
                next_time = time.time()
    except KeyboardInterrupt:
        pass
    finally:
        self.running = False
        self._publish_status(retain=True, state="STOPPED")
        try:
            self.mqtt.loop_stop()
            self.mqtt.disconnect()
        except Exception:
            pass
: (省略)
```