SEN0670 串口通信协议

一、物理层

项目	参数
接口	UART2（SerialWE2）
引脚	PB9=UART2_RX, PB10=UART2_TX（在部分板型上受 PB11 模式选择影响）
默认波特率	9600
数据位	8
停止位	1
校验位	无
流控	无
电平	3.3V TTL
发送方式	UART DMA + TX 环形缓冲（36KB），单次 DMA chunk 最多 4095 字节
接收方式	UART DMA 单字节回调入 RX 环形缓冲
行结束符	`\r` 或 `\n`
最大命令长度	4096 字节（`SSCMA_CMD_MAX_LENGTH`）

支持波特率：110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 230400, 460800, 921600

> 说明：PB9/PB10 是否处于 UART2 模式在部分板型上由 PB11 电平和板级配置决定，请以设备启动日志与实际固件配置为准。

二、AT 命令输入格式

2.1 基本语法

```
AT+命令名[=参数1,参数2,...]<CR/LF>
```

- 前缀必须是 `AT+`。
- 命令名大小写不敏感，内部会转大写匹配。
- 支持路由标签：`AT+Tag@CMD=...`，以最后一个 `@` 后的部分作为实际命令名。
- 字符串参数使用双引号，例如：`AT+POSELEARN="Posture_1"`。

2.2 参数匹配规则（按当前源码）

- 参数个数严格按命令注册参数签名匹配，个数不匹配会直接报参数错误。
- `POSEFINISH` / `HANDFINISH` 当前实现需显式带参数，建议使用：
  - `AT+POSEFINISH=0` 或 `AT+POSEFINISH=1`
  - `AT+HANDFINISH=0` 或 `AT+HANDFINISH=1`
- `POSELEARN` 当前可靠语法为单参数名称：`AT+POSELEARN="name"`。
  - 虽然内部回调兼容读取第二参数位，但当前注册签名只允许 1 个参数。

2.3 响应格式

当前固件存在两类 JSON 回包风格。

1. 带 `code_text` / `ok`（`at_reply_prefix()` 路径）：

```
{"type":0,"name":"ID?","code":0,"code_text":"EL_OK","ok":1,"data":"3343029e"}
```

2. 精简 JSON（`main_task.hpp` 手工拼接，常见于学习和串口配置命令）：

```
{"type":0,"name":"BAUDRATE","code":0,"data":115200,"message":"Baudrate set successfully, reboot required to take effect"}
```

另外，AT 解析错误使用 `type=2`：

```
{"type":2,"name":"AT","code":5,"code_text":"EL_EINVAL","ok":0,"data":{"message":"..."}}
```

完整串口包固定格式：

```
\r + JSON正文 + \n
```

完整包示例（线上的实际边界）：

```
\r
```

```
{
  "type": 0,
  "name": "ID?",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": "3343029e"
}
```

```
\n
```

> 说明：除 `HELP?`（纯文本）外，JSON 回包均遵循以上 `\r ... \n` 边界；文档中的多数示例为便于阅读，仅展示 JSON 正文。

2.4 输出协议模式

模式	设置命令	说明
JSON 文本（默认）	`AT+TPROTO=0`	文本 JSON 输出
二进制	`AT+TPROTO=1`	0x55AA 帧封装（见 §五）

三、错误码

code	常量	含义
0	EL_OK	成功
1	EL_AGAIN	可重试
2	EL_ELOG	逻辑错误
3	EL_ETIMOUT	超时
4	EL_EIO	IO/存储错误
5	EL_EINVAL	参数非法
6	EL_ENOMEM	内存不足
7	EL_EBUSY	设备忙
8	EL_ENOTSUP	不支持
9	EL_EPERM	当前状态不允许
10	EL_FAILED	通用失败

四、AT 命令对（请求 -> 响应）

> 说明：本章若示例只展示 JSON 正文，串口线上仍是完整包 `\r + JSON + \n`。

4.0 总览表

分类	命令
设备信息	HELP?, ID?, NAME?, STAT?, VER?
设备控制	RST, BREAK
模型	MODELS?, MODEL, MODEL?
算法	ALGOS?, ALGO, ALGO?
传感器	SENSORS?, SENSOR, SENSOR?
推理	INVOKE, INVOKE?
串口配置	BAUDRATE, BAUDRATE?, TPROTO, TPROTO?, TPKTSZ, TPKTSZ?, TKPTS, TKPTS?
算法阈值	TSCORE, TSCORE?, TIOU, TIOU?, TSIMILARITY, TSIMILARITY?
姿态学习	POSELEARN, POSELEARN?, POSEFINISH, POSESTOP, POSELIST?, POSERENAME, POSEREMOVE, POSECLEAR, POSEAUTOLEARN
手势学习	HANDLEARN, HANDLEARN?, HANDFINISH, HANDSTOP, HANDLIST?, HANDRENAME, HANDREMOVE, HANDCLEAR, HANDAUTOLEARN

4.1 设备信息

4.1.1 HELP? - 获取帮助

- 请求：`AT+HELP?`
- 响应：纯文本（非 JSON）

4.1.2 ID? - 获取设备 ID

请求：`AT+ID?`


```
{"type":0,"name":"ID?","code":0,"code_text":"EL_OK","ok":1,"data":"3343029e"}
```


4.1.3 NAME? - 获取设备名称

请求：`AT+NAME?`


```
{"type":0,"name":"NAME?","code":0,"code_text":"EL_OK","ok":1,"data":"DFRobot
Human Pose"}
```


4.1.4 STAT? - 获取设备状态

请求：`AT+STAT?`


```
{"type":0,"name":"STAT?","code":0,"code_text":"EL_OK","ok":1,"data":
{"boot_count":42,"is_ready":1}}
```


4.1.5 VER? - 获取版本信息

请求：`AT+VER?`


```
{
  "type": 0,
  "name": "VER?",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": {
    "at_api": "v0",
    "software": "2026.04.02",
    "hardware": "v1.0.0"
  }
}
```

4.2 设备控制

4.2.1 RST - 重启设备

- 请求：`AT+RST`
- 说明：设备直接复位，通常不会收到完整成功回包。

4.2.2 BREAK - 停止当前任务

请求：`AT+BREAK`

```
{
  "type": 0,
  "name": "BREAK",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": 1234567
}
```

`data` 为停止时刻毫秒时间戳。

4.3 模型

4.3.1 MODELS? - 获取可用模型

请求：`AT+MODELS?`

```
{
  "type": 0,
  "name": "MODELS?",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": [
    {"id": 1, "type": 3, "address": 4194304, "size": 123456},
    {"id": 2, "type": 3, "address": 4325376, "size": 234567}
  ]
}
```

4.3.2 MODEL=ID - 设置模型（挂起切换）

请求：`AT+MODEL=<id>`

```
{
  "type": 0,
  "name": "MODEL",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": {
    "model": {"id": 1, "type": 3, "address": 4194304, "size": 123456}
  }
}
```

> 说明：`MODEL` 命令阶段主要设置 pending 状态，真正装载在后续 `INVOKE` 准备阶段执行。

4.3.3 MODEL? - 获取当前模型

请求：`AT+MODEL?`

```
{
  "type": 0,
  "name": "MODEL?",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": {"id": 1, "type": 3, "address": 4194304, "size": 123456}
}
```

> 注意：`MODEL?` 的 `data` 是模型对象本体，不是 `data.model`。

4.4 算法

4.4.1 ALGOS? - 获取可用算法

请求：`AT+ALGOS?`

```
{
  "type": 0,
  "name": "ALGOS?",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": [
    {"type": 3, "categroy": 1, "input_from": 1},
    {"type": 2, "categroy": 0, "input_from": 1}
  ]
}
```

4.4.2 ALGO=ID - 设置算法

请求：`AT+ALGO=<type_id>`

```
{
  "type": 0,
  "name": "ALGO",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": {"type": 3, "categroy": 1, "input_from": 1}
}
```

4.4.3 ALGO? - 获取当前算法

请求：`AT+ALGO?`

```
{
  "type": 0,
  "name": "ALGO?",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": {"type": 3, "categroy": 1, "input_from": 1}
}
```

> 注意：当前 AT 回包不输出 `config` 字段。

4.5 传感器

4.5.1 SENSORS? - 获取可用传感器

请求：`AT+SENSORS?`

```
{
  "type": 0,
  "name": "SENSORS?",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": [
    {
      "id": 1,
      "type": 1,
      "state": 1,
      "opt_id": 0,
      "opt_detail": "QVGA 320x240",
      "opts": {
        "0": "QVGA 320x240",
        "1": "VGA 640x480"
      }
    }
  ]
}
```

> 注意：`opts` 是对象映射（key 为字符串化 option id），不是数组。

4.5.2 SENSOR=ID,EN,OPT - 设置传感器

请求：`AT+SENSOR=<id>,<enable>,<opt_id>`

```
{
  "type": 0,
  "name": "SENSOR",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": {
    "sensor": {"id": 1, "type": 1, "state": 1, "opt_id": 0, "opt_detail": "QVGA
320x240"}
  }
}
```


4.5.3 SENSOR? - 获取当前传感器

请求：`AT+SENSOR?`


```
{
  "type": 0,
  "name": "SENSOR?",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": {
    "sensor": {"id": 1, "type": 1, "state": 1, "opt_id": 0, "opt_detail": "QVGA
320x240"}
  }
}
```

4.6 推理控制

4.6.1 INVOKE=N,DIFFERED,RESULT_ONLY - 启动推理

请求：`AT+INVOKE=<n_times>,<differed>,<result_only>`

参数	类型	取值	说明
n_times	int	`-1` 或正整数	推理次数，`-1` 表示持续运行直到 `BREAK`
differed	int	0/1	0 每帧回包，1 仅结果变化时回包
result_only	int	0/1	0 带图像，1 不带图像

首条响应（type=0）：

```
{
  "type": 0,
  "name": "INVOKE",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": {
    "timestamp_ms": 1000,
    "model": {"id": 1, "type": 3, "address": 4194304, "size": 123456},
    "algorithm": {"type": 3, "categroy": 1, "input_from": 1},
    "sensor": {"id": 1, "type": 1, "state": 1, "opt_id": 0, "opt_detail": "QVGA 320x240"}
  }
}
```


流式事件（type=1，目标检测示例）：


```
{
  "type": 1,
  "name": "INVOKE",
  "code": 0,
  "data": {
    "count": 12,
    "seq": 12,
    "timestamp_ms": 123456,
    "image_base64_len": 21876,
    "perf": [8, 21, 3],
    "boxes": [[87, 83, 77, 65, 70, 1]],
    "resolution": [640, 480],
    "image": "<BASE64_JPEG>",
    "rotate": 0
  }
}
```


流式事件（type=1，姿态关键点示例，`TKPTS=1`）：


```
{
  "type": 1,
  "name": "INVOKE",
  "code": 0,
  "data": {
    "count": 3,
    "seq": 3,
    "timestamp_ms": 10086,
    "image_base64_len": 0,
    "perf": [10, 40, 2],
    "pose_keypoints": [
      [
        [100, 120, 80, 160, 92, 1],
        [[120, 50, 95, 0], [130, 55, 90, 1]]
      ]
    ],
    "resolution": [640, 480],
    "pose_class": {
      "available_classes": [{"id": 1, "name": "Posture_1"}],
      "learning": {
```

```
        "state": 0,
        "state_name": "idle"
      }
    }
  }
}
```

流式事件（type=1，手势关键点示例，`TKPTS=1`）：

```
{
  "type": 1,
  "name": "INVOKE",
  "code": 0,
  "data": {
    "count": 4,
    "seq": 4,
    "timestamp_ms": 10123,
    "image_base64_len": 0,
    "perf": [7, 18, 2],
    "hand_keypoints": [
      [
        [150, 120, 90, 90, 88, 1],
        [[160, 90], [165, 95], [170, 100]]
      ]
    ],
    "resolution": [640, 480],
    "hand_class": {
      "available_classes": [{"id": 1, "name": "Gesture_1"}],
      "learning": {
        "state": 0,
        "state_name": "idle"
      }
    }
  }
}
```

`INVOKE` 忙碌响应：

```
{
  "type": 0,
  "name": "INVOKE",
  "code": 7,
  "code_text": "EL_EBUSY",
  "ok": 0,
  "data": "busy"
}
```

4.6.2 INVOKE? - 查询推理状态

请求：`AT+INVOKE?`

未运行：

```
{
  "type": 0,
  "name": "INVOKE?",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": 0
}
```

运行中：

```
{
  "type": 0,
  "name": "INVOKE?",
  "code": 0,
  "code_text": "EL_OK",
  "ok": 1,
  "data": 1
}
```

> 当前实现返回状态位 `0/1`。源码中"最近一次 type=1 缓存回放"路径目前未启用。

4.7 串口配置

4.7.1 BAUDRATE=VAL - 设置波特率

请求：`AT+BAUDRATE=<value>`

支持值：110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 230400, 460800, 921600

成功示例：

```
{
  "type": 0,
  "name": "BAUDRATE",
  "code": 0,
  "data": 115200,
  "message": "Baudrate set successfully, reboot required to take effect"
}
```

> 修改后需重启生效。

4.7.2 BAUDRATE? - 查询当前波特率

```
{
  "type": 0,
  "name": "BAUDRATE?",
  "code": 0,
  "data": 9600
}
```

4.7.3 TPROTO=MODE

- 请求：`AT+TPROTO=<mode>`
- `mode`：`0`=JSON, `1`=binary

4.7.4 TPROTO?

```
{"type":0,"name":"TPROTO?","code":0,"data":0}
```

4.7.5 TPKTSZ=SIZE

- 请求：`AT+TPKTSZ=<size>`
- 范围：`32..1024`

4.7.6 TPKTSZ?

```
{"type":0,"name":"TPKTSZ?","code":0,"data":192}
```

4.7.7 TKPTS=MODE

- 请求：`AT+TKPTS=<mode>`
- `mode`：`0`=关键点折叠为框，`1`=输出关键点（默认）

4.7.8 TKPTS?

```
{"type":0,"name":"TKPTS?","code":0,"data":1}
```

4.8 算法阈值

4.8.1 TSCORE=VAL

```
{"type":0,"name":"TSCORE","code":0,"code_text":"EL_OK","ok":1,"data":60}
```

4.8.2 TSCORE?

```
{"type":0,"name":"TSCORE?","code":0,"code_text":"EL_OK","ok":1,"data":60}
```

4.8.3 TIOU=VAL

```
{"type":0,"name":"TIOU","code":0,"code_text":"EL_OK","ok":1,"data":45}
```

4.8.4 TIOU?

```
{"type":0,"name":"TIOU?","code":0,"code_text":"EL_OK","ok":1,"data":45}
```

4.8.5 TSIMILARITY=VAL

{"type":0,"name":"TSIMILARITY","code":0,"code_text":"EL_OK","ok":1,"data":80}

4.8.6 TSIMILARITY?

{"type":0,"name":"TSIMILARITY?","code":0,"code_text":"EL_OK","ok":1,"data":80}

> 说明：在 `INVOKE` 运行期间，阈值命令可能由运行时命令表接管，回包字段风格可能是精简 JSON。

4.9 姿态学习（Pose Learning）

4.9.1 POSELEARN="NAME" - 开始学习

请求：`AT+POSELEARN="<name>"`

{"type":0,"name":"POSELEARN","code":0,"data":1}

> 当前注册签名仅 1 参数。`AT+POSELEARN="name",1` 在当前解析器下不可靠，不建议使用。

4.9.2 POSELEARN? - 查询学习状态

请求：`AT+POSELEARN?`


```
{
  "type": 0,
  "name": "POSELEARN?",
  "code": 0,
  "data": {
    "learning": {
      "state": 1,
      "state_name": "collecting",
      "collected": 5,
      "min_samples": 10,
      "frames": 6,
      "max_frames": 100,
      "accepted_samples": 5,
      "rejected_samples": 1,
      "quality_score": 84,
      "missing_target_streak": 0,
      "missing_target_streak_limit": 10,
      "multiple_persons_streak": 0,
      "multiple_persons_streak_limit": 15,
      "reject_reason_last": "low_kp_quality"
    }
  }
}
```


结束阶段可能附带字段（按条件出现）：

- `result`：`"completed"` / `"failed"`
- `conflict`：`{"id":1,"name":"Posture_1","sim":0.923,"threshold":0.800}`

- `error` / `error_name` / `message`

4.9.3 POSEFINISH=FORCE - 完成学习

请求：`AT+POSEFINISH=<force>`

- `force=0` 正常结束
- `force=1` 强制结束

```
{"type":0,"name":"POSEFINISH","code":0,"data":1}
```

4.9.4 POSESTOP - 停止学习

```
{"type":0,"name":"POSESTOP","code":0,"data":1}
```

4.9.5 POSELIST? - 已学姿态列表

```
{"type":0,"name":"POSELIST?","code":0,"data":["Posture_1","Posture_2"]}
```

4.9.6 POSERENAME=ID,"NEW_NAME"

```
{"type":0,"name":"POSERENAME","code":0,"data":1}
```

4.9.7 POSEREMOVE="NAME"

```
{"type":0,"name":"POSEREMOVE","code":0,"data":1}
```

4.9.8 POSECLEAR

```
{"type":0,"name":"POSECLEAR","code":0}
```

4.9.9 POSEAUTOLEARN - 自动命名并启动学习

```
{"type":0,"name":"POSEAUTOLEARN","code":0,"data":{"pose_name":"Posture_1"}}
```

> 说明：该命令只负责启动学习会话并返回自动名称。学习完成由后续 `INVOKE` 流中的自动收敛逻辑决定。

4.10 手势学习（Hand Learning）

4.10.1 HANDLEARN="NAME"

```
{"type":0,"name":"HANDLEARN","code":0,"data":1}
```

4.10.2 HANDLEARN?

```
{
  "type": 0,
  "name": "HANDLEARN?",
  "code": 0,
```

```
  "data": {
    "learning": {
      "state": 1,
      "state_name": "collecting",
      "collected": 4,
      "min_samples": 10,
      "frames": 5,
      "max_frames": 100,
      "accepted_samples": 4,
      "rejected_samples": 1,
      "quality_score": 81,
      "missing_target_streak": 0,
      "missing_target_streak_limit": 10,
      "reject_reason_last": "low_kp_quality"
    }
  }
}
```


4.10.3 HANDFINISH=FORCE


```
{"type":0,"name":"HANDFINISH","code":0,"data":1}
```


4.10.4 HANDSTOP


```
{"type":0,"name":"HANDSTOP","code":0,"data":1}
```


4.10.5 HANDLIST?


```
{"type":0,"name":"HANDLIST?","code":0,"data":["Gesture_1","Gesture_2"]}
```


4.10.6 HANDRENAME=ID,"NEW_NAME"


```
{"type":0,"name":"HANDRENAME","code":0,"data":1}
```


4.10.7 HANDREMOVE="NAME"


```
{"type":0,"name":"HANDREMOVE","code":0,"data":1}
```


4.10.8 HANDCLEAR


```
{"type":0,"name":"HANDCLEAR","code":0}
```


4.10.9 HANDAUTOLEARN


```
{"type":0,"name":"HANDAUTOLEARN","code":0,"data":{"hand_name":"Gesture_1"}}
```


> 说明：同 `POSEAUTOLEARN`，该命令是启动学习，不是单条命令内完成全流程。

五、二进制协议（TPROTO=1）

5.1 通用帧头（11 字节）

```
Offset  Size  Field        Description
0       1     sof0         0x55
1       1     sof1         0xAA
2       1     ver          0x01
3       1     msg_type     消息类型
4       1     flags        bit0=1 末分片
5       2     seq          uint16 LE
7       2     payload_len  uint16 LE
9       2     crc16        CRC16-CCITT (0x1021, init=0xFFFF)
```

CRC 计算范围：帧头前 9 字节 + payload。

5.2 AT 回复封装（msg_type=0x11）

payload = `at_rsp_hdr + data_bin`

```
at_rsp_hdr (13 bytes):
format_ver(1)=2
resp_type(1)=JSON的type
code(2)=JSON的code
cmd_id(2)=命令ID
reserved(2)=0
data_len(4)=data_bin长度
```

`data_bin` 值类型：

- `BIN_NULL(0x00)`
- `BIN_BOOL(0x01)`
- `BIN_I64(0x02)`
- `BIN_U64(0x03)`
- `BIN_F64(0x04)`
- `BIN_STRING(0x05)`
- `BIN_OBJECT(0x06)`
- `BIN_ARRAY(0x07)`
- `BIN_BYTES(0x08)`

分片：单帧 payload 超过 `TPKTSZ` 时自动切片。

5.2.1 成功回包压缩规则（当前实现）

下列命令在 `type=0 && code=0` 时，二进制 AT 封装可进行 data 压缩（`data_len=0`）：

- `MODEL`
- `ALGO`
- `SENSOR`
- `BAUDRATE`
- `TPROTO`
- `TPKTSZ`
- `TKPTS`
- `TSCORE`
- `TIOU`
- `TSIMILARITY`

5.3 INVOKE 二进制流消息类型

msg_type	名称	说明
`0x00`	INVOKE_META	元信息/错误
`0x01`	INVOKE_BEGIN	帧开始
`0x03`	DET_CHUNK	检测框分片
`0x04`	HAND_KP_CHUNK	手关键点分片
`0x05`	POSE_KP_CHUNK	姿态关键点分片
`0x06`	IMAGE_CHUNK	图像分片（base64 字节）
`0x07`	INVOKE_END	帧结束

`IMAGE_CHUNK` 元数据包含 `rotate_deg`（0/90/180/270）。

5.4 上位机解析步骤

1. 扫描 `0x55 0xAA`
2. 读取 11 字节头并检查 `payload_len`
3. 读取 payload 并校验 CRC16
4. `msg_type=0x11` 按 AT 回复解包；`0x00..0x07` 按 INVOKE 流解包
5. 对未知 `cmd_id` 做兼容兜底处理

六、典型调试流程

```
1. AT+ID?
2. AT+VER?
3. AT+STAT?
4. AT+MODELS?
5. AT+MODEL=1
6. AT+ALGOS?
7. AT+ALGO=3
8. AT+INVOKE=-1,0,0
9. AT+BREAK
```

七、常见问题

现象	可能原因	排查
无响应	传输模式或引脚模式不符	确认当前板型 PB11 模式选择及 UART2 连线
乱码	波特率不匹配	先尝试 9600，再切到目标波特率
`code=7`	INVOKE 正在运行	先发 `AT+BREAK`
`INVOKE?` 一直是 1	推理任务持续运行	`AT+BREAK` 后重查
无图像字段	使用了 `result_only=1`	改为 `AT+INVOKE=-1,0,0`
无关键点坐标	`TKPTS=0`	改为 `AT+TKPTS=1`

> 版本：v2.1 | 日期：2026-05 | 基于当前仓库 `sscma` 源码实现（as-is）