Polar Bridge项目总述

该项目的开发过程目前可大致分为五个环节。以下列出各环节的任务执行情况与遇到的难点和解决方案。

一、基础框架搭建

本轮完成的任务（按实际推进顺序）

A. 电脑端-录制质量校验工具（Step 0 的一部分）

- 把原先的 `qa_check.py` 改造成可视化选文件的 `check_xdf.py`（加入文件对话框、中文提示、规则阈值可配）。
- 修正 numpy/array 判真值报错（把 `if not ts:` 改为安全长度判断）。
- 输出清晰：流清单、时长、标记列表与计数、结论（PASS/NOT PASS）+ 告警。

迭代轮次（约）： 10
关键决议：

- 使用文件对话框而不是硬编码路径，任何人都能直接跑。
- 保留“密度告警但不拉红”策略（不影响 PASS/FAIL）。
- 术语统一、避免“黑话”，脚本打印使用直白中文。

B. UDP-LSL 通道构建与日志

- 拆分**数据通道**与**标记通道**：
  - 数据：`UdpSender`（心跳/数据流）。
  - 标记：`MarkerBus → UdpMarkerBridge → UDPSenderService`（仅发 marker）。
- 给 `UDPSenderService` 加状态/错误/发送日志（host:port、ready、send ok/err、payload）。
- 解决“看不到桥日志/连接拒绝”等问题（确保与“应用设置”统一目标；在 `onAppear` 触发桥接订阅）。

迭代轮次（约）： 8
关键决议：

- **强日志**是排错利器：打印 state、payload、错误码。
- “应用设置”时**同时更新**数据与标记的 UDP 目标，防止两路目标不一致。

C. UI 总体架构与导航

- 建立项目目录结构与共享配置 `AppConfig`。
- 首页（HomeView）+ 采集页（CollectView）骨架 + 调试页（DebugView）。
- **移除底部 TabBar**（避免多入口造成困惑），统一通过首页卡片导航进入各页。
- 首页设备卡（Verity/H10）设计与空态文案、布局打磨（对齐、间距、色弱友好）。

对话轮次（约）： 12
关键决议：

- 主页“设备 → 任务 → 历史”三段式信息架构。
- 去 TabBar、保留**单入口**路径：更符合实验现场操作心智。
- “最近数据”改成**历史记录**卡片（TaskRow 风格，暂时占位）。

D. 采集页四区骨架

- **A 区｜采集状态**：试次、被试 ID、采集状态、标记数 —— 统一等宽 chip，可横向滑动、可扩展。
- **B 区｜数据选择**：按已连接设备动态生成 DataPill（PPI/PPG/HR/ECG），可选/可取消；空态清晰。
- **C 区｜采集操作**：大按钮（开始/停止）+ 计时；下方 5 个标记按钮。
- **阶段计时**：总时长 + 基线/诱导/干预三段；采用**“按需显示”**策略（只显示已开始的阶段），避免按钮高度剧烈变化。
- **禁用/顺序/高亮**：未开始禁用；必须按“基线→诱导→结束→干预→结束”顺序；当前激活高亮。
- **D 区占位**：暂留“根据实验需求继续开发”（以后接均值心率、RMSSD 等）。

迭代轮次（约）： 25（这一块最多）
关键决议：

- 阶段计时采用**增量显示**（你选的 B 方案）：避免 UI 抖动。
- **顺序管控**落在 `AppStore`：`markerSequence + markerStep + canEmit/emitMarkerInOrder`。
- **禁用规则**与**可用条件**都由 Store 推导，UI 只做展示，不自揣逻辑。

E. 状态统一归口 `AppStore` + 线程安全（@MainActor）

- 把心跳/循环/开始停止等**实际逻辑迁到 AppStore**（单一真相源），调试页只调用 Store 方法。
- `AppStore` 顶层加 `@MainActor`，规避“后台线程发布 UI 变更”警告。
- 需要后台回调时，用 `Task { @MainActor in … }` 切回主线程（如在 UDP 回调/定时器闭包中更新 @Published）。

迭代轮次（约）： 10
关键决议：

- **只保留一个开关**：`isCollecting`（合并了 isSending 概念）。
- Store 负责：开始/停止、计数、计时、选择、标记推进；页面只绑定。
- UDP 发送目标用 `@AppStorage` 持久化，三页一致。

F. 端到端链路验证 & 回归

- 完善 iPhone → UDP → `udp_to_lsl.py` → LSL → LabRecorder → XDF 全任务流程并测试。
- `check_xdf.py` 校验：PASS（数据时长、标记条数、标签计数都正确）。
- 多次真实演练（基于 31s 案例：基线 ~7s、诱导 ~6s、干预 ~5s），脚本与 UI 时序一致（±1s 可接受）。

迭代轮次（约）： 3
关键决议：

- 1s 粒度的计时/UI 与 LSL/XDF 的时间差在 ±1s 属合理（UI 刷新 + 网络与工具采样差）。
- 以后接入真实 Verity/H10 数据后，再把 D 区做成“轻统计”。

重要问题与解决方案（摘录）

- **不能看到桥日志 / 发送被拒** → 为 `UDPSenderService` 加“重建连接 + 状态日志 + payload 打印”，并在“应用设置”同时更新数据/标记通道的目标，修复“目标不一致”。
- **Swift 编译/类型推断卡住** → 拆表达式、用小常量（如 `availableSorted`、`buttonTitle`、`elapsedText`），避免在 View builder 里做复杂推导。
- **主线程发布警告** → `@MainActor` 到 AppStore 顶层，必要处用 `Task { @MainActor in … }` 包裹。
- **设备状态与可订阅信号两套口径** → 统一由 `verityState/h10State` 推导 `availableSignals`，调试页的 Picker 改这两个枚举状态即可联动 B 区。
- **导航混乱** → 删除 TabBar，只保留首页 → 采集/调试/历史 三个**单入口**。

每块任务的迭代轮次（大约）

模块	轮次（约）
Step 0 + XDF 质检脚本改造	~10
底层 UDP/桥接/日志	~8
UI 架构与导航（T0/T1/T2）	~12
采集页四区 + 阶段计时（T4-1～T4-7）	~25
AppStore 归口与线程模型	~10
端到端联调 + 回归（T9）	~3
合计	**~68**

> 说明：是“粗略分段计数”，为帮助回顾精力投入的主要区间。

形成的原则性决议（后续继续沿用）

1. **单一真相源**：所有状态与动作归口 `AppStore`；页面不自带逻辑。
2. **强日志，可追踪**：网络/桥接/标记都打印关键路径与错误码。
3. **一条主线入口**：去 TabBar，避免多入口造成决策成本与歧义。
4. **低干扰 UI**：阶段计时“按需出现”，不挤压主按钮；D 区先留白，等真实数据再定义指标。
5. **术语直白**：文案与脚本输出避免黑话，必要术语配解释。
6. **先通路、再加值**：先把“线插上”打通闭环，再补统计/可视化/可访问性。

结语

这轮把**从脚本校验 → 底层通道 → UI 框架 → 功能闭环 → 端到端验收**全跑通了，而且每个薄弱环节（日志、线程、导航、顺序约束）都加了“防呆”。按这个基座，下一步直接进入 **Polar SDK 接入**（替换心跳模拟为真实 PPI/PPG/HR/ECG），基本不动 UI，就能看到“真数据上屏、进 XDF、QA 通过”的完整流水线。

二、功能框架跑通

本轮完成的任务（按实际推进顺序）

① 搭建 H10 基础链路（多轮）

- 目标：从 Polar H10 获取真实生理信号，并以 UDP→LSL→LabRecorder 全链路录制。
- 过程与产出：
  - 用 PolarBleSdk 6.5.0 重写 `PolarManager`：扫描、连接、订阅心率 HR 与 RR（新 API `startHrStreaming`；正确解析 `PolarHrData` 批）。
  - 修正 observer 注册、协议适配、主线程隔离（`@MainActor` / `DispatchQueue.main`）与 RxSwift `Disposable` 生命周期。
  - 通过 `CollectView` 的“药丸”选择驱动订阅；在控制台与 UI 显示设备、连接与数据状态。
- 决议：HR 与 RR 必须**所见即所得**，不做自动捆绑；UI 以你定义的 `SignalKind` 为准。

② 扩展 ECG/ACC（多轮）

- 目标：把 H10 的 ECG(130 Hz) 与 ACC(可选采样率/量程)流纳入，同步发 UDP。
- 过程与产出：
  - `requestStreamSettings(.ecg/.acc)` 获取设置集合，选择 MVP 组合（ECG 130 Hz；ACC 50 Hz ±4 g），批量打包发送，附带 `seq` 、`n` 、`t_device`。
  - 日志中验证批量大小与数值范围合理；ACC 用于后续运动伪迹判定。
- 决议：ECG/ACC 采用**批量 JSON**，不单点发包，减轻 UDP 压力，便于下游缓冲。

③ UDP→LSL 桥接脚本与网络发现（多轮）

- 目标：把 iPhone 发的 UDP 文本桥接为 LSL 流，供 LabRecorder 录制。
- 过程与产出：
  - `udp_to_lsl.py`：创建数据与标记两路 LSL outlet，周期 summary 打印。
  - Zeroconf 广播 `_pbudp._udp`。修正优先选择 RFC1918 私网地址，避免 169.254/198.18 虚拟网段。
  - 用 `dns-sd` 逐步定位 mDNS/Bonjour 行为。
- 决议：由于 iOS 本地网络权限、mDNS 环境复杂，**放弃自动发现**作为默认路径；以“**手工设置 UDP 目标**”为主线，自动发现当作“锦上添花”（可关可开）。

④ iOS 侧 UDP 目标配置与 UI（多轮）

- 目标：用户能在 App 内**显式指定** UDP 目标 IP:Port，并可随网络切换。
- 过程与产出：
  - 新增统一弹窗体系（Sheet）、键盘工具栏、输入合法性检查，`AppStorage` 持久化；首页顶部“设置 UDP”卡片，实时显示当前目标。
  - 去掉早期“本机网段信息”误导项；所有界面一致化（TaskRow 的 A/B 模式合并，保留灰色卡片样式）。

- 决议：**以人为本**，优先可控性与可见性；自动化发现不是 MVP 必要条件。

⑤ 事件标记链路（控制流）（多轮）

- 目标：在实验中标注“基线/诱导开始/诱导结束/干预开始/干预结束”。
- 过程与产出：
  - `MarkerBus` + `UdpMarkerBridge` 常驻；UI 五个按钮发 Marker。
  - LabRecorder 录得第二路 `PB_MARKERS_TEST`，`qa_check.py` 验证标记数量与标签名，PASS。
- 决议：先不扩展更多标签按钮，避免 UI 误触；五个足够覆盖段落边界。

⑥ 统一数据规范（JSON UDP 载荷）与 Telemetry 文档（数轮）

- 目标：所有流的 JSON 字段、类型、时间戳、批量结构统一，便于解析与下游分析。
- 过程与产出：
  - 定稿：
    - HR：`{"type":"hr","bpm":Int,"t_device":Double,"device":"H10"}`
    - RR：`{"type":"rr","ms":Int,...}`
    - ECG：`{"type":"ecg","fs":130,"uV":[...],"seq":Int,"n":Int,...}`
    - ACC：`{"type":"acc","fs":50,"range_g":4,"mG":[[x,y,z],...],"seq":Int,"n":Int,...}`
  - 明确 `t_device` 为 iPhone 系统时间，**锁时交给 LSL**。
- 决议：MVP 不在 iOS 端做时钟对齐与高精度时间戳；端到端延迟与偏差控制交由 LSL 的时钟同步。

⑦ XDF→CSV/图形/质检脚本（多轮）

- 目标：把录得的 XDF 拆解为 CSV、绘图，并产出 QA 文本报告。
- 过程与产出：
  - 解决 numpy/pandas ABI 与 GUI 后端问题（Agg），加入目录选择器。
  - 生成 `ecg.png/hr.png/rr.png`，叠加 Marker；`qa_report.txt` 含采样完整性、时间一致性、HR 与 RR 的一致性、事件对齐的烟雾测试。
- 决议：Python 作为分析主线；R 排除；Matlab/SPSS/Stata 作为可选补充。

⑧ Windows 迁移与多设备同步策略（讨论并定向）（少量轮次）

- 目标：把桥接与录制放到 Windows；新增呼吸带（USB/COM）接入。
- 过程与产出：
  - 方案定稿：Windows 上运行 UDP→LSL 脚本与 LabRecorder；呼吸带在本机产生数据后推 UDP 或直接发 LSL，由 LabRecorder 锁时；手机端 Marker 与生理流同样走 UDP→LSL，**统一在 PC 端对齐**。

- 决议：同步点在 LSL；避免跨端时钟；先做稳定与可控的“单 PC 汇聚”。

⑨ Verity 支持策略（决策）

- 目标：引入 Verity Sense 的 PPG/PPI。
- 决议：不做“设备能力自描绘”以免大改；按**经验映射**在 `availableSignals` 放出 `.ppg/.ppi/.hr/.acc`，在 `PolarManager` 只新增两段 `startPpgStreaming/startPpiStreaming`，与现有 JSON 口径一致。

本轮难题（按难度排序）

A. mDNS/Bonjour 自动发现不稳定（最难，跨任务）

- 出现场景：任务③，网络发现与自动填充 UDP 目标；也影响任务④的 UI 逻辑与用户体验。
- 核心难点：
  - iOS 本地网络权限、`NSBonjourServices`、`NSLocalNetworkUsageDescription` 缺一不可。
  - 多网卡/虚拟网卡（Tailscale/VMware 等）让 Zeroconf 选到 169.254/198.18 等“假”地址。
  - 局域网中的 mDNS 组播被家用路由/AP 或运营商设备“优化/屏蔽”。
- 尝试过的方案与失败原因：
  1. 端到端 Zeroconf（iOS 查找 `_pbudp._udp`，Python 广播）：在部分网络环境下只收到 169.254/198.18；iOS 端解析不到或解析到错误地址。
  2. `dns-sd` 三段式排查（注册/浏览/解析）：能定位问题，但不能保证用户现场永远“绿灯”。
  3. Python 端强制挑选首个 `A` 记录：在多接口机器上经常不是想要的 RFC1918。
- 最终方案：
  - Python 端 Zeroconf 广播**保留**，但强制**优先使用 RFC1918 私网 IPv4**，并降级策略（仅作为“可选提示”）。
  - iOS 端**不依赖自动发现**；以“**手动设置 UDP 目标**”为主流程。
- 迭代：贯穿数十轮；最终定论是“工程现实优先，手设可控，自动发现非刚需”。

B. Polar SDK API 细节与类型坑（高难，任务①②）

- 出现场景：从 HR/RR 开始，到 ECG/ACC 设置与批处理。
- 核心难点：
  - `startHrStreaming` 返回的是**批**（数组）且 `PolarHrData` 是**元组数组**；不能用 `.hr` 直取单值。
  - 旧 API `deviceHrObserver` 已废弃；必须用 `startHrStreaming`。

    - `requestEcgSettings` 并不存在；统一用 `requestStreamSettings(identifier, feature: .ecg/.acc/...)`。
- 尝试与失败：
    - 早期按旧文档调用，编译错误或运行时拿不到值。
- 最终方案：
    - 统一 RxSwift 链式，`onNext` 遍历批次，结尾样本作“当前 HR”，RR 用 `flatMap` 全量发；ECG/ACC 以 settings→start→批处理→批量 JSON 发送。
- 迭代：多轮，逐点修正 API 与类型。

C. 主线程隔离与 SwiftUI 驱动（中高难，任务①④⑤）

- 出现场景：在数据回调里读取/写入 `AppStore`（`@MainActor`），导致 “Main actor-isolated property cannot be referenced”。
- 核心难点：iOS 蓝牙/流回调常在后台队列，SwiftUI/`@Published` 必须在主线程更新。
- 尝试与失败：
    - 直接访问 `AppStore.shared` 属性与方法，编译器报并发错误。
- 最终方案：
    - 回调尾部用 `DispatchQueue.main.async { … }` 或把相关方法标注 `@MainActor`；`Disposable` 生命周期在 `PolarManager` 内自洽。
- 迭代：少数轮次，明确规范后一次性修正。

**D. LLDB/断点与异常暂停

- 出现场景：点击 ECG 启动时跳入 `lldb`，显示 Breakpoint/constraint 日志。
- 核心难点：Xcode 默认“异常断点”“UIViewAutoLayout 警告”会强行中止流。
- 尝试与失败：
    - 误以为代码崩溃，反复排查。
- 最终方案：
    - 清理异常断点；保留控制台告警；将布局冲突修复为非阻断。
- 迭代：一次。

E. UI 一致性与弹窗系统（中等，任务④⑤）

- 出现场景：TaskRow 双模式（Button/NavigationLink）导致“灰色卡片背景”时有时无；弹窗有的挂在 ScrollView，有的嵌在行内。
- 核心难点：SwiftUI 的修饰器作用域与 List/Navigation 的默认背景。
- 尝试与失败：
    - 行内 sheet 导致空白页；A/B 模式切换丢样式。
- 最终方案：

    - 统一成“卡片样式 + 外部 sheet 调用 + 键盘工具栏”；把 UDP 与受试者信息都走同一弹窗系统，保留你期望的“灰卡片”外观。
- 迭代：多轮，逐步收敛。

F. UDP 发送异常与“连接被拒绝”（中等，任务③④）

- 出现场景：早期 iPhone 仍发向“旧的家庭网络”IP；Python 报 `Connection refused`。
- 根因：目标 IP 未更新；或 UDP 服务未监听；或路由切换。
- 最终方案：
    - 在 App 顶部显式显示并可编辑目标 IP:Port，改完立即 `recreateConnection`；在 Python 端打印来源 IP 与计数，便于核对。
- 对话：中等轮次，问题伴随网络发现一并解决。

G. Python 分析环境（中等，任务⑦）

- 出现场景：`pandas` / `numpy` 的 ABI 冲突、GUI 后端阻塞。
- 方案：锁定 `python -m pip install -U numpy pandas matplotlib pyxdf`；脚本强制 `matplotlib.use('Agg')`；用 Tk 目录选择器；自动保存图片与 `qa_report.txt`。
- 结果：可在 IDE 直接运行，不依赖命令行参数。
- 迭代：少量轮次。

H. 能力自描绘（低优先，任务⑨讨论）

- 议题：是否用 `requestStreamSettings` “探测即驱动 UI”。
- 取舍：考虑固定设备（H10/Verity）与稳定性要求，**保留经验映射**；探测仅作 DEBUG 校验，不驱动 UI。
- 对话：若干轮，最终压缩为“以后再说”。

结语

这次我们把“手机采集 Polar 生理信号 → UDP → 电脑端 LSL → LabRecorder → XDF/CSV/图像/QA 报告”的**端到端路径**跑通并固化：UI 清晰可控，SDK 用法正确，数据规范统一，标记可溯源，分析脚本能画能检。自动发现这类“锦上添花”不再阻塞主线；Verity 支持走最小改动面即可。你现在就能按实验流程稳定采集基线/诱导/干预三段数据，后处理与统计分析也有抓手。

三、核心功能优化

本轮完成的任务（按实际推进顺序）

① 方向澄清与总体方案

- 澄清了“掉线/重连”指的是 Wi-Fi/UDP—not Polar 蓝牙；确认“iOS 发 UDP、Python 侧做 LSL 翻译”的架构是对的。
- 评估了校园网导致分片/乱序/丢包的风险，结论：**做 UDP 载荷“控包”**（应用层切分），保留原有“最小可行”传输路径，由设置页开关控制。

② 设置页与特性开关

- 新增 `SettingsView` 与导航按钮；落地 `FeatureFlags`（如 `cappedTxEnabled`、`progressLogEnabled`、`verboseLogs`、`maxPacketBytes` 可调）。
- `CollectView` 受设置开关控制是否显示“采集进度卡”。

③ UDP 发送通路梳理

- 修复 **UDPSenderService 未初始化目标地址** 导致电脑端收不到数值流的问题（在 AppStore 绑定阶段就 `applyTarget()`）。
- 用 `tcpdump` 验证包长，指导如何看是否可能触发 IP 层分片。

④ 应用层“控包”实现

- 为 **PPG、ACC、ECG、PPI** 建立统一的 **按最大字节数切分** 的发送函数（保持 JSON schema 不变），每段计算 `t_device`、独立 `seq`；关闭开关时走旧路径。
- 引入 **切包事件统计**：`capEvents`（PM 内发送）、`capStats`（AppStore 聚合 min/max/avg/count），在 **Information/Progress** 卡中按需显示。

⑤ “Already in state” 根因与治理

- 观察到偶发 `gattAttributeError(errorCode: 6, "Already in state")`。
- 第一次缓解：**串行化操作队列**（每设备 `opQueueByDevice` + `opSerial`），避免并发 start/stop。
- 最终方案：把“**冷却时间**”升级为“**就绪回执**”——只有当 **流真正开始产出数据（onNext）** 或确认幂等成功，才视为完成，放行下一条；同时 **队列去重**，避免重复 `start(kind)` 入队。
- 扩展：断连/全停时清空队列、当前项与处理状态，防止僵尸状态。

⑥ 日志降噪与定位

- 加入 `vlog` 受 `verboseLogs` 控制；保留关键摘要（applySelection 前后差分、队列推进、start/stop 成功/失败等）。

⑦ RR/PPI vs HR 的时序解释与分析口径

- 解释“RR/PPI 比 HR **整体更早**”的两个来源：
  1. **BeatEventAligner** 用 RR/PPI 反推上一个心跳的时间（自然比 HR 回调早）；

2. Verity 的 HR 是聚合/滤波结果，回调在后。

- 建议**比较口径**：把 HR 当参考，在窗口内对齐 RR/PPI（或把 RR/PPI 还原为拍点 `t_e` 与 HR 比较），而不是逐点同刻值对比。

⑧ 文档与图标

- 更新 README：在 iOS 总览中加入“控包+排队传输”的目的与机制，并补充“代码结构与职责”。
- 产出一枚 1024×1024 的极简“脉搏/心形”风格图标样稿。

> 大致迭代轮次：方向与风险评估（~10+）、设置与 FeatureFlags（~8）、UDP 初始化与收数问题（~6）、控包实现与统计（~12）、“Already in state”定位与两轮治理（~25+）、tcpdump 验证（~5）、RR/PPI 对齐与分析口径（~6）、文档与 UI 收尾（~8）、杂项与 Xcode/iPhone 连接波动排查（~8）。

本轮难题（按难度排序）

A. Polar 流 “Already in state”（最高难度；跨任务反复出现）

- **出现位置**：开始采集（PPG/PPI 等 start）阶段，且偶发。
- **核心难点**：
  - Polar SDK 的流状态机是“**边发命令边产出**”；如果上一次流没**完全就绪/完全停止**，立刻再来同类命令，就会回 “已经在该状态”。
  - 我们的 UI、ApplySelection、订阅回调、以及“连线恢复/服务重启”的竞态，容易在边界触发**重复 start**。
- **尝试过的方案**：
  1. **固定冷却时间**（失败/不稳）：时间窗选不好，要么慢，要么仍撞车。
  2. **把 onSubscribe 当成功回执**（不充分）：订阅建立≠设备进入产流稳定态，仍有窗口。
  3. **只在错误出现时吞掉**（风险）：把 “Already in state” 当成功，会掩盖真实的双发问题。
- **最终方案**（采用）：
  - **每设备串行队列 + 去重**：同类 `start(kind)` 若已在队列/正在处理，则丢弃；
  - **就绪回执**：以 **onNext 首包**（或明确幂等）作为“动作完成”的信号，再放行下一条；
  - **断连/全停清理**：清空队列、当前项与处理标记，防止残留。
- **为什么可行**：把“命令节流”从**时间**变为**状态**，让状态机只在**已确认稳定**后推进，天然抹掉竞态。
- **迭代轮次**：定位与第一次串行化（~10+）、复现与第二轮“就绪回执/去重”（~15+）。

B. UDP 控包 + 避免 IP 分片 的整体可靠性（中高难度）

- **出现位置**：iOS→Python UDP→LSL 的整链路。
- **核心难点**：
  - MTU/以太网/802.11 的实际承载不等，同一 JSON 模式下各流（PPG/ACC/ECG/PPI）载荷差异大；
  - 既要控包，又要**尽量保持时间/排序**与 LSL 的一致。
- **尝试过的方案**：
  - 单纯“分片数量固定/粗略估算”→在不同流/不同时刻仍会超阈。
- **最终方案**：
  - 统一的 **二分逼近切段**（以 `maxPacketBytes` 为上限），每段自带 `t_device` 与递增 `seq`；
  - **统计回传**到 UI，便于观测阈值是否需要调整；
  - `tcpdump` 验证：在 “length < ~1472(UDP负载阈)” 区间活动，极少触发 IP 分片。
- **迭代轮次**：设计与实现（~12）、验证与统计（~6）。

C. UDPSenderService 未设目标 导致“只收标记没数值”（中难度）

- **出现位置**：切包改造后，电脑端只收到 marker。
- **核心难点**：
  - `UDPSenderService` 的目标地址原先在设置页“保存”时才 `applyTarget()`；直接运行/独立启动时未绑定。
- **最终方案**：
  - 在 `AppStore.bindPolar()`/初始化阶段就同步目标；
  - 统一所有发送走 `UDPSenderService.shared.send(...)`。
- **对话轮次**：排查与修复（~6）。

D. Xcode ↔ iPhone 调试链路 波动（中等难度，环境问题）

- **出现位置**：偶发 “Lost connection / timeout / tunnel…”
- **核心难点**：macOS/iOS/Xcode 版本、网络调试（Connect via Network）、证书/信任、线缆等外因。
- **方案**：线缆直连、关闭“经网络连接”、重启设备服务、信任重置。
- **对话轮次**：建议与观测（~8）。

E. RR/PPI 与 HR 的时间错位（中等难度，方法学）

- **出现位置**：CSV 对齐与可视化校验。
- **核心难点**：
  - RR/PPI 是“上一个拍点”的估计，HR 是聚合/滤波的率；
  - `BeatEventAligner` 与 LSL 时钟映射叠加，出现“整体更早”。
- **最终方案与口径**：

    - 用 **拍点时间** `t_e` 对齐 HR 的时间轴做窗口一致性检验，或延长测量让自然重叠；
    - 避免“同刻值逐点比较”的误用。
- **对话轮次**：解释与口径调整（~6）。

F. 日志与 UI 观测（较低难度）

- **出现位置**：排障期日志过多、难以阅读；需要在信息页展示切包统计。
- **方案**：
    - `vlog` + `verboseLogs` 总开关；
    - `capEvents` → `capStats` →Information/Progress 卡按需显示（仅采集中且开关打开）。
- **对话轮次**：实现与微调（~8）。

结语

这轮我们把**可靠传输**（控包）与**可靠启动**（队列+就绪回执）两条主干都扎牢了：前者尽量避免网络侧“随机性”，后者把设备侧“状态机竞态”压平。加上 UI 的可观测（统计/日志开关）、README 的可传达、以及 tcpdump 的可验证，现在这套桥接基本达到**可用 + 可诊断 + 可迭代**的状态。后续若再遇到偶发表现，有了这些“钩子”，定位和收敛速度会快很多。

四、提升功能稳定性

本轮完成的任务（按实际推进顺序）

A. 打通 Verity 的连接与状态管理（约 5 次迭代）

- **做了什么**：从 HomeView 点击设备卡片触发 AppStore.tapDeviceCard → PolarManager.connect；为 Verity 完成连接路径，补全 `DeviceState`（连接中/已连/可连），让 HomeView 正确“点亮”与切换状态。
- **关键决议**：不把 `startPpi(...)` 塞进 `startHR()`，而是以信号选择为核心的“选择→应用到 PM → 再分别开流”的管线；Verity 和 H10 的 HR/ACC 在枚举上分开命名（VHR/VACC vs HHR/HACC），避免语义混淆。

B. “可订阅信号”到“选择信号”的闭环（约 4 次迭代）

- **做了什么**：连接后由 PM `probeCapabilities()` 得到 `availableSignals`；AppStore 生成带前缀的 `availableSignalsLabeled`；CollectView 用它渲染“选择数据”卡片；用户选择后更新 `selectedSignals` 并回灌到 PM；PM `applySelection()` 分设备、分类型独立开/关在线流。
- **关键决议**：修正“只连 Verity 时仍显示 RR”的错误来源（历史上 RR 是随 H10 HR 流的一个维度），从源头（PM 与标签）区分清楚。

C. Verity 流：ACC/PPG/PPI 的细化与稳定（约 6 次迭代）

- **做了什么**：
  - **ACC**：修复 `gattAttributeError: Invalid Number of channels/Resolution`，显式带上 `.channels=3`，参数以 `requestStreamSettings` 返回的集合挑选（52Hz、±8G、16bit）。
  - **PPG**：确认可用设置是 `fs=55Hz / res=22bit / 4ch`，打印批处理批次 `batch n=...`；解释 55Hz 为设备特性而非 bug。
  - **PPI**：解释“启动后延迟数十秒才出数据”为 Polar 算法回溯与质量评估的机制（非代码问题）；在 iOS 侧透传 `ppErrorEstimate / blockerBit / skinContactStatus / skinContactSupported`。
- **关键决议**：**不**强行把 PPI 提前为“秒级响应”，而是把延迟作为实验 SOP 提前告知；同时在数据侧补充“beat 级时间戳（te）”，为后续锁时与跨流对齐提供原语。

D. H10 流：ECG/ACC/HR/RR 订阅与参数策略（约 4 次迭代）

- **做了什么**：
  - **ECG**：130Hz / 14bit；
  - **ACC**：查询可用设置（25/50/100/200Hz；±2/±4/±8G；16bit），默认 50Hz/±4G；
  - **HR/RR**：HR 流内独立拆出 RR 间期，展开为“单事件”。
- **关键决议**：ACC 采样率**不是越高越好**，研究常用 25–50Hz 足以评估运动伪迹；高采样在低功耗 BLE 下会挤占带宽并增大抖动。

E. 多LSL 数值流桥接-建立Hub（约 8 次迭代）

- **做了什么**：
  - 保留“极简转发”的 `udp_to_lsl.py`（文本流 + markers）。
  - 新建 `bridge_hub.py` 作为**唯一需要运行**的入口：自带 UDP 监听 + 文本/标记 LSL outlet + **装载翻译器**。
  - 新建 `Translators/polar_numberic.py`：把来自 iOS 的 JSON 包（ECG/ACC/PPG/HR/RR/PPI）拆到**多路数值 LSL**（按设备命名），并在 LSL metadata 里落会话与参数。
  - 增加用户态提示、摘要计数、异常打印、NAME_SUFFIX 开关、路径导入修复。
- **关键决议**：**解耦**：保持 `udp_to_lsl.py` 纯转发；把“认识包结构、创建数值流”的逻辑放在可热插拔的翻译器里，未来其它设备（皮电/肌电/脑电）各写各的 translator，不污染基础桥接层。

F. QA/可视化工具链（约 10 次迭代）

- **做了什么**：

- `check_xdf.py`（后改名 `polar_check_xdf.py`）用于**结构核查**：是否出现期望的多路数值流，采样数/时长/采样率范围是否合理。修复了 numpy array → float 报错、旧版回退解析文本流逻辑、命名同步问题。
- `polar_xdf_to_csv.py` 做**结构化导出与报告**：分设备分类型 csv（含单位），PPG/ACC 的 fs/通道含义、HR 的 bpm、PPI 的 ms/quality/flags，**追加 te** 字段。提供清晰的汇总报告。
- `polar_data_plot_validity.py` 做**内容体检**：
  - PPG：completeness 与四通道一致性；
  - ACC：高运动占比；
  - HR vs PPI：MAE 与 bias；
  - RR/PPI：时序（tachogram）与 Poincaré；
  - ECG：原始轨迹（含 50/60Hz 干扰解读指南）。
  - 给出 PASS/WARN/FAIL 标尺与清洗建议。

- **关键决议**：检验/导出**限定 Polar**，建立“设备-针对性”的translator (例如 `polar_numbric.py`)；外设进来就写新 translator + 新 QA，以“插件化”保证可维护性。

G. 双设备并联与资源管理（约 6 次迭代）

- **做了什么**：
  - PolarManager 重构为**按 deviceId 管理**的订阅集合， disposables 与 seq 各自隔离，`applySelection()` 按设备生效；
  - 连接/断开支持“点击同一张卡片切换”，避免全局 `connectingId` 带来的竞态；
  - `stopAllStreams()` 逐设备安全停止；
  - 修复 ACC/PPG/HR/PPI/RR 在多设备情况下混发/误发的问题。
- **关键决议**：不再使用全局 wantHR / wantRR”这类跨设备开关，所有流与状态都**绑定 deviceId**。

H. 实测与解读（约 4 次迭代）

- **做了什么**：完成 Verity: PPG+PPI+VHR；H10: ECG+RR+HHR 的整套实验；用 QA 与图形解读“PPI 慢一拍”、PPG 55Hz 批处理、ECG 里的工频刺点与肌电抖动、ACC 运动片段等。
- **关键结论**：**科研优先级**一般为 H10（ECG/RR 金标准思路，RR 稳定；ACC 足够），Verity（PPG/PPI 可用，注意 PPI 延迟与质量阈值，PPG 宜配清洗流程）。两者并联可做“跨模态验证”。

本轮难题（按难度排序）

(1) 多设备并联的状态/流管理

- **产生位置与迭代次数**：跨任务：A/B/G/E，约 6–8 次
- **难点**：原始逻辑有“全局 connectingId / wantHR”影子，切换设备时互相“踢线”；订阅/停止与 UI 选择耦合不清，多源时常发生“流开在了错的设备上”。
- **尝试**：
  1. 补丁式地继续使用全局标志（**失败**：竞态不可控、边界太多）；
  2. 局部地把流按类型拆，但仍混用全局 seq/Disposable（**失败**：仍有串流与复用错误）；
  3. **最终方案**：**所有状态（activeStreams、disposables、seqs）都以 deviceId 为 key**，`applySelection(id)` 仅影响此 id，connect/disconnect 也仅改自己的桶。
- **原因**：BLE 多外设就是“多上下文”，全局状态迟早打架。
- **结果**：Verity+H10 能稳定并联，UI 点击任意卡片可独立连/断。

(2) UDP→LSL 的分层与可扩展设计

- **产生位置与迭代次数**：任务 E，约 6–8 次
- **难点**：要做到“**只运行一个脚本**”同时又不把所有设备逻辑塞成巨型脚本；要解决 pylsl 版本 API 差异（`resolve_stream`）、包路径导入、LabRecorder 的流在线性等问题。
- **尝试**：
  1. 全靠 `udp_to_lsl.py` 做所有事情（**失败**：功能臃肿且难以泛化）；
  2. 两个脚本让用户先后运行（**被否**：操作复杂且易错）；
  3. **最终方案**：`bridge_hub.py` **单入口**，内置文本/markers outlet，并按需加载 `Translators/polar_numberic.py`（可拓展更多 translator）。
- **结果**：Hub 启动即完成 LSL 基础通道搭建，手机一开流就会动态创建数值 LSL；用户只用跑一个脚本。

(3) Verity ACC/PPG 的 GATT/设置错误

- **产生位置与迭代次数**：任务 C，约 3–4 次
- **难点**：`Invalid Number of channels`、`Invalid Resolution` 报错。
- **尝试**：沿用 H10 参数（**失败**），硬编码通道数/分辨率（**不稳**）。
- **最终方案**：严格以 `requestStreamSettings` 的返回集合为准，**显式带上** `.channels=3` 与合法的 `resolution`，并对 fs/range 从集合里择优。
- **结果**：Verity ACC 正常；PPG 也以 55Hz/22bit/4ch 工作。

(4) PPI 启动延迟与“慢一拍”

- **产生位置与迭代次数**：任务 C/H，约 2–3 次
- **难点**：用户期待“点击即有”，但 PPI 是算法估计 + 质量评估，有**回溯延迟**与**批量**。
- **尝试**：调整订阅时序与“激活集”宣告（**表症改善，但非根因**）。
- **最终方案**：接受设备机理，**文档化延迟**；把 **te（事件时间）** 落入包与 CSV，供后续锁时/对齐与质量分析使用。

- **结果**：实验 SOP 可预期；HR vs PPI 的 MAE/bias 有量化与可视解释。

(5) Lab Recorder 报“红色流/离线”

- **产生位置与迭代次数**：任务 E，约 2–3 次
- **难点**：流名存在但显示离线；有时是本机多网卡/IPv6 responder 警告，有时是 LabRecorder 自身状态残留。
- **尝试**：强行指定 IP、关 Bonjour（**影响不大**）。
- **最终方案**：
  - 复位流程（关掉终端/App/LabRecorder 全部重启）、且 Hub 打印**明确本机监听 IP**与“下一步操作提示”；一般可恢复。
  - LabRecorder 中订阅流与开始操作不要连续进行，先用手机开始采集十几秒，停止后更新 LabRecorder 检查是否识别到LSL流。全部选择后，再开始手机采集，再从 LabRecorder 点击开始。
- **结果**：现场操作可控。

(6) CSV 导出报错（ndarray→float）与命名同步

- **产生位置与迭代次数**：任务 F，约 3–4 次
- **难点**：不同流的 `time_series` 既有 1D 也有嵌套；改名后 plot 脚本找不到文件。
- **尝试**：逐 case 显式判断（**易漏**）。
- **最终方案**：写通用 `flatten_1d()`；CSV 命名统一为 `PB_<type>_<device>.csv` 并在 `polar_data_plot_validity.py` 侧适配 `glob` 模式。
- **结果**：导出稳定；可视化能自动找到当次导出的命名。

(7) RR 数值流“看不见”

- **产生位置与迭代次数**：任务 E/F，约 2 次
- **难点**：iOS 已发 RR 包，Hub 没创建 RR outlet → LabRecorder 没看到。
- **原因**：`polar_numberic.py` 未覆盖 RR 分支
- **最终方案**：在 translator 新增 RR 路由与 outlet 创建、推送；同步在导出脚本中增加 RR te 列。
- **结果**：RR 数值流出现，CSV 也有 te。

(8) 路径与依赖问题（pylsl、包导入）

- **产生位置与迭代次数**：任务 E，约 2–3 次
- **难点**：`resolve_stream` 不同版本差异、`Libs` / `Translators` 导入失败。
- **方案**：改用 `resolve_byprop` 或仅 `StreamInlet`；启动时手动把根目录/子目录加到 `sys.path`；在子目录加 `__init__.py`。
- **结果**：环境跨平台可用。

(9) UI/代码回归（sfSymbol、applyParticipant 丢失）

- **产生位置与迭代次数**：跨任务 A/B/G，约 2 次
- **难点**：多处大改后，一些 UI 辅助属性与参与者信息接口短暂失联。
- **方案**：按**最小改动原则**恢复旧接口签名与位置；对 UDP 注入的内容不作“猜写”。
- **结果**：已手动恢复缺失点，后续遵循“外部接口尽量不改名”的约束。

共性问题（核心难点与结论）

- **“多设备 → 全局状态”**：一切“全局开关/全局序号/全局订阅”最终都会打架；**按 deviceId 分桶**是结构性答案。
- **“桥接层 → 业务层”**：把“认识数据语义”的逻辑塞到桥接会不可维护；**Hub + Translator 插件化**是通用解。
- **“设备机理 → 用户预期”**：PPI 非即刻；PPG 55Hz 非低配；RR/HR 来自不同路径；**在工具链与文档中提前设计 SOP 和质量度量**，比“强行优化到违背机理”更可持续。
- **“命名与元数据”**：**统一命名 + 落单位/参数**（fs/range/res/ch 等）大幅降低跨会话调试成本。

结语

以上是整段合作的任务闭环与难题纵览。现在这条链路从**iOS（多设备多流）→ UDP → Hub（文本/标记 + 数值翻译）→ 多路 LSL → XDF → CSV → 可视 QA**已经打通，并且关键的“PPI 延迟/beat 时间轴”“多设备并联”“Verity/H10 参数差异”“LabRecorder 在线性”等坑都给出了工程化落地与文档化 SOP。后续如果加皮电/肌电/脑电，只需要**新增一个 translator + 一个 QA 脚本**，主桥与现有 Polar 工具都无需重写。

五、功能风险检验

本轮完成的任务（按实际推进顺序）

A. 仓库审读与基础答疑（迭代≈4）

扫描 PolarBridge 与 UDP2LSL，确认 H10 ECG 默认 130 Hz、Telemetry 报文含 `seq`，评估 125–130 Hz 在心理生理里是否够用；给出 MVP 功能合理性与缺口清单。

B. 电脑端 UDP 丢包/抖动统计（迭代≈12）

在 `bridge_hub.py` 中接入 `udp_metrics.py`，实现每流 loss/rate/jitter/gap60s 的滚动统计与 JSONL 落盘；修正日志路径到 `UDP2LSL/logs/<会话>/`；加简报打印口径。

C. Ping/Pong 时钟与延迟轨迹（迭代≈10）

iOS 侧加 `PingPongResponder.swift`，Python 侧 `ping_pong.py`；最初收不到数据，原因是 ping 包缺 `device` 字段与走了废弃通道，改为 `UDPSenderService` 广播并附 `device`；随后把 ping/pong 从丢包统计里排除，避免污染 loss。

D. 质量报告与可视化（迭代≈8）

`udp_packet_quality_report.py` 读取 `metrics.jsonl` 画 RTT、ECG loss、gap 率；后来加图例/线型区分与“怎么读图”的人话说明。三轮实验对比：家庭 Wi-Fi（大包）→ 缩到 800B → iPhone 热点 600B，得出“热点最稳；包越小越少丢”的决议。

E. LSL 镜像录制设计与落地（迭代≈20+，过程最曲折）

方案定为**LSL 级镜像**：`lsl_mirror.py` 实时写 Parquet；CSV 转换移到独立 `mirror_parquet_to_csv.py`。期间踩坑：

- 把镜像嵌进 `bridge_hub` 导致耦合与卡死 → 决议“**彻底解耦，两个进程分别跑**”。
- 终止与导出混在一起、SIGTERM 不稳 → 决议“**镜像只负责写 Parquet，按 ESC 退出；CSV 另跑**”。
- Parquet→CSV 命名与主线不一致 → 统一为 `*_kind_device.csv`。
  最终达成：镜像稳定写入；CSV 转换与主线命名一致。

F. 主线 vs 镜像一致性对比（迭代≈9）

重写校验思路：先**流覆盖一致性**，再用 **Markers（baseline/stop）对齐** 或自动重叠窗口，比较统计特征，最后做**时间快检**（互相关滞后为主，极值时间差做参考）。一次对比中 ECG/ACC 被“极值差”误报，结论是保留为 INFO，不当 FAIL。

G. 文档与操作流程更新（迭代≈4）

把“UDP 网络质量评估”“镜像录制”两块写入 README 的电脑端章节；重写“采集流程”“数据检查”小节，明确路径规范、脚本顺序、可选步骤与判读口径。

H. 一键流程可行性评估（迭代≈2）

结论：用**跨平台 Python orchestrator** 串 `udp_packet_quality_report → polar_xdf_to_csv → mirror_parquet_to_csv(可选) → compare(可选) → plot_validity` 最稳；难点在“会话匹配与统一交互”，可通过入口脚本一次性解决。

- ⚠️：此阶段未实现

本轮难题（按难度排序）

① LSL 镜像的生命周期与退出一致性

- **产生位置与迭代次数**：出现在任务 E，迭代≈20+，跨任务牵连 B/C/F
- 难点：和 `bridge_hub` 进程耦合、终止信号不一致、导出与录制交织、异常路径导致“有索引没数据”。
- 尝试方案：
  1. 在 `bridge_hub` 内托管镜像子进程（失败：阻塞、清理复杂、卡死风险大）。
  2. 镜像内集成 Parquet→CSV（失败：退出竞态，CSV 时机不受控，易丢文件句柄）。
- 最终方案：**完全解耦**。`lsl_mirror.py` 只写 Parquet，按 **ESC** 退出；CSV 由 `mirror_parquet_to_csv.py` 事后显式执行；命名与主线对齐。原因：把职责缩到最小，去掉 90% 不确定性。

② 会话与路径匹配

- **产生位置与迭代次数**：任务 G/H，迭代≈4
- 难点：`logs/<会话>`、`main_lsl_data/<会话>`、`mirror_lsl_data/<会话>` 命名来源不同，时间不一致，用户容易找错。
- 失败做法：让各脚本弹窗各自问，结果交互割裂。
- 最终方案：规范路径与命名，文档明确；后续用 orchestrator 让用户只选一次 XDF，其他自动匹配并打印“我猜你想要这个会话”。

③ Ping/Pong 收不到与误计入丢包

- **产生位置与迭代次数**：任务 C，迭代≈10
- 难点：初期 `device` 未携带、走了废弃 `UDPSender`，导致 PC 端不识别；随后 Ping/Pong 混入 metrics 让 loss 失真。
- 方案：改由 `UDPSenderService` 发含 `device` 的 ping，Python 端按 `type in {ping,pong}` 直接跳过统计。

④ 网络质量与可视化的可读性

- **产生位置与迭代次数**：任务 D，迭代≈8
- 难点：指标太多用户读不懂；线条颜色/图例不清。
- 方案：固定三图（RTT、ECG loss、gap 率），给“好/警告/不建议”的人话阈值；图例与线型区分；结合实测提出“热点优先 + 包限 600–900B”的可操作建议。

⑤ 主线 vs 镜像的时间对齐误报

- **产生位置与迭代次数**：任务 F，迭代≈3
- 难点：极值时刻差对尖峰过敏，把噪声当错位。

- 方案：以互相关滞后 `lag_ms` 为主判，极值差仅信息参考；必要时对极值前做轻度平滑/赢泽化。

⑥ 路径/依赖与平台差异

- **产生位置与迭代次数**：任务 B/D/E/G/H，迭代≈6
- 难点：日志默认到 `~/lsl_logs`、mac 沙箱、防火墙、`tkinter` 缺失、路径含空格。
- 方案：统一改到仓库内 `UDP2LSL/logs` / `Data/...`，一律 `pathlib`，GUI 不可用时走命令行参数。

结语

总之，这轮把“**能看见的网络质量**”和“**可自救的镜像备份**”两件最要命的事落地了：你现在能定位丢包、给出可复现实验条件；主线崩了也有镜像兜底。剩下的一键化只是体力活，用 orchestrator 把交互统一就行。没糖衣，只有能用的东西。