DiPECS 中期进展报告

——端侧 AIOS 原型从概念到可运行系统的跨越

中期进展报告

项目名称：DiPECS

作者单位：DiPECS Team

报告类型：中期进展报告

完成时间：2026 年 4 月 27 日

目录

1 引言 **3**

2 项目定位 **3**

3 系统闭环 **4**

4 平台选择 **5**

5 架构分层 **5**

6 机制与策略分离 **6**

7 中期成果 **7**

7.1 里程碑状态 7

7.2 基础采集原型 9

7.3 Rust 核心链路 9

7.4 测试与验证 10

8 研究背景与问题动机 **10**

8.1 移动端体验痛点 10

8.2 语义鸿沟 10

9 技术路线 **11**

9.1 Android 信息来源 11

9.2 多源信息融合 11

9.3 资源调度策略 12

10 当前局限 **12**

11 后续计划 **13**

12 结论 **13**

DiPECS 中期进展报告

——端侧 AIOS 原型从概念到可运行系统的跨越

摘要： **目的：**汇报 DiPECS 中期阶段的核心进展，验证 Rust 核心类型体系、隐私空气隙、窗口聚合、规则式意图生成、策略校验、动作执行骨架和离线测试框架是否已形成可运行链路。**方法：**以 Android 13（API 33）为中期开发基线，采用 Rust + Kotlin 工程路线，通过单元测试、离线 replay 和 Android 公开 API 原型验证各模块行为。**结果：**中期已完成核心类型、隐私脱敏、窗口聚合、规则决策、策略校验、动作总线、动作执行骨架和 golden trace/replay 测试框架；Android 侧完成了 UsageStats、通知和基础系统状态的采集验证；数据流已具备可脱敏、可聚合、可决策、可审查、可回放能力。**结论：**项目已跨越从概念到可运行原型的阶段，下一阶段需优先打通 Android 到 Rust 的生产入口、完善 golden trace/replay、接入真实云端后端，并验证低风险动作是否带来可测系统收益。

关键词： 中期进展；Android 端侧系统；事件驱动架构；隐私空气隙；机制与策略分离

1 引言

DiPECS 是一个面向 Android 端侧环境的意图预测系统原型。其核心命题是：在用户动作实际发生之前，系统能否根据近期上下文预测用户可能需要的资源，并提前进行低风险准备，从而降低冷启动或跨应用切换时的可感知延迟。

中期汇报时，项目目标平台设定为 Android 13（API 33）和可定制 AOSP 环境；实现语言采用 Rust + Kotlin。Rust 负责核心逻辑，包括事件 schema、隐私脱敏、上下文聚合、策略校验、动作状态机和离线回放；Kotlin 负责 Android 应用层，包括权限、服务、系统 API 调用和用户界面。需要说明的是，该目标平台为中期阶段的开发基线；结题验证已迁移至 Android 15（API 35）模拟器环境，详见结题报告。

项目不把云端 LLM 视为直接控制设备的执行者。LLM 或本地规则只能输出结构化建议，例如 intent、confidence、risk 和 suggested actions；本地 Rust 策略模块保留最终执行权。这是中期阶段最重要的系统边界。

2 项目定位

DiPECS 的研究定位可概括为三点：第一，它是端侧上下文 AIOS 原型，而不是单纯的 LLM 应用；系统关心的是采集、脱敏、聚合、决策、策略和审计之间的边界。第二，其默认数据源是现实可部署的数据源：Android 侧主要使用公开 API 生成 append-only

JSONL；Linux daemon 侧使用本机进程和系统状态；离线 replay 使用相同 schema 验证管线行为。第三，它把动作安全置于模型能力之前：决策后端默认本地规则优先，云端 LLM 只有在环境变量和配置完整时才参与；所有动作建议都必须经过策略引擎和 `ActionLifecycle`，最终形成可审计的终态记录。

3 系统闭环

中期阶段定义的系统闭环为：本地采集 → 隐私脱敏 → 上下文聚合 → 意图预测 → 策略校验 → 本地优化执行 → 结果记录。表 1 对各阶段进行了说明。

表 1: 中期阶段定义的系统闭环

阶段	内容
本地采集	通过 Android 公开 API 和系统状态接口采集应用切换、通知、时间、网络、电量等信号。
隐私脱敏	将通知文本、位置、文件路径等敏感信息转成粗粒度标签或摘要，避免 raw data 进入决策层。
上下文聚合	将连续事件按时间窗口聚合为 StructuredContext，形成可输入模型或规则的统一上下文。
意图预测	由规则基线或云端模型输出候选 intent、置信度、风险等级和建议动作。
策略校验	PolicyEngine 根据风险、置信度、能力等级和动作类型判断是否允许执行。
本地执行	只执行低风险优化，如 KeepAlive、PreWarm / Prefetch 的安全子集或记录 hint。
结果记录	通过 trace、audit record 和 golden replay 验证行为是否可复现、可审计。

4 平台选择

中期阶段选择 Android 13 作为目标平台，主要基于稳定性和可验证性：

(1) Android 13 的公开 API 相对成熟，UsageStatsManager、NotificationListenerService 和基础设备状态接口足以验证第一阶段算法闭环[1;2]；

(2) AOSP 环境便于后续从普通 APK 原型逐步下沉到 root、daemon 或 framework 层，测试更强的资源调度能力；

(3) Rust 交叉编译和 Android NDK 工具链已经可以支持核心库复用，降低后续 Kotlin/Rust 混合开发风险；

(4) Android 12 之后的系统具备 eBPF 基础能力，为后续内核级事件观测保留扩展空间[3]。

因此，中期设计采用渐进路线：先在普通应用权限内验证采集、脱敏和策略闭环；再在更高权限环境中评估真实资源收益。

5 架构分层

中期阶段完成了对 Rust workspace 的分层梳理，目标是避免上层业务逻辑污染底层 schema 和策略边界。表 2 给出了主要模块的职责。

表 2: 中期阶段的工程分层

层级	模块	职责
协议层	aios-spec	定义 RawEvent、SanitizedEvent、StructuredContext、Intent、Action 和 trace schema。
核心层	aios-core	实现隐私空气隙、窗口聚合、策略引擎、动作生命周期和 trace 验证逻辑。
采集层	aios-collector	接入 Android JSONL、本机系统采集、进程/系统状态读取和采集统计。
决策层	aios-agent	提供 RuleBased、Cloud/Mock 后端和 DecisionRouter。
动作层	aios-action	接收 AuthorizedAction，提供离线 adapter 和默认执行器骨架。
工具层	aios-cli / aios-daemon	提供 replay、pipeline、daemon runtime 和观测入口。

这套分层体现了机制与策略分离。云端或 agent 层可以提出建议，但 schema、policy 和 action lifecycle 位于本地核心层，负责最终边界控制。

6 机制与策略分离

中期汇报中强调了“云端负责 policy-like 推理，本地保留 mechanism 执行权”的原则。更准确地说，模型层负责意图理解和候选动作生成；本地核心负责确定性审查和执行。表 3 从多个维度对比了两层职责。

表 3: 机制与策略分离

维度	云端/决策后端	本地 Rust core
性质	非确定性推理或规则判断	确定性状态机和策略校验
职责	意图理解、模式推断、候选动作生成	脱敏、上下文聚合、权限与风险校验、审计
状态	可失败、可降级、可替换	可回放、可测试、可追责
安全边界	只输出结构化建议	保留最终授权和执行权

该原则避免了让 LLM 直接调用 Android API 或系统命令。即使模型输出了动作，本地也会根据风险等级、置信度、动作白名单和上下文 target 检查决定是否授权。

7 中期成果

7.1 里程碑状态

表 4 汇总了中期阶段主要里程碑的完成情况。

表 4: 中期阶段主要里程碑

里程碑	状态	说明
核心事件类型与 schema 定义	完成	`RawEvent`、`SanitizedEvent`、`StructuredContext`、`IntentBatch` 等类型已定型。
`PrivacyAirGap` 隐私脱敏	完成	敏感字段不进入后续后端，回归测试覆盖泄漏场景。
`WindowAggregator` 窗口聚合	完成	事件序列可聚合为统一上下文。
RuleBased 规则意图生成	完成	本地规则基线可输出候选意图。
`PolicyEngine` 策略校验	完成	风险、置信度、capability、target 检查已落地。
`ActionLifecycle` 动作执行骨架	完成	proposal、verdict、dispatch、终态审计链路已通。
`ActionBus` 动作总线	完成	采集与处理异步解耦。
GoldenTrace / replay 框架	骨架完成	数据结构与离线 replay 方向已确定，待扩充数据集并纳入回归。
Android 公开 API 采集验证	完成	UsageStats、NotificationListener、设备状态已原型验证。
Android action bridge	原型中	动作执行骨架已具备，真实 bridge 待结题阶段强化。
云端 LLM 后端	接口设计	已定义 `CloudLlmBackend` 与回退路径，真实云端接入待后续完成。
eBPF / Binder 高权限采集	后续阶段	当前仅留扩展接口，不作为中期默认目标。

7.2 基础采集原型

Android 侧已经围绕公开 API 验证基础采集方向：

- 使用 UsageStatsManager 采集应用使用和前后台切换信息；
- 使用 NotificationListenerService 采集通知来源、类别、channel 和出现/撤销事件；
- 采集基础设备上下文，如时间、网络、电量和屏幕状态[4;5]；
- 通过 append-only JSONL 或等价 trace 形式，将 Android 事件转入 Rust 侧处理链路。

这些采集源可以支撑第一阶段用户意图判断。中期阶段尚未把 Accessibility、Binder trace 或 eBPF 作为主路径，而是保留为后续增强方向[6]。

7.3 Rust 核心链路

中期阶段 Rust 侧已经完成最小闭环所需的主要模块：

- **事件模型**：在 aios-spec 中定义 RawEvent、SanitizedEvent、StructuredContext、IntentBatch、SuggestedAction 和 trace 相关类型；
- **隐私脱敏**：PrivacyAirGap 将原始事件转换为脱敏事件，确保通知正文、文件路径等敏感字段不会进入后续后端；
- **窗口聚合**：WindowAggregator 将事件按时间窗口组织为 StructuredContext，并生成 ContextSummary；
- **意图生成**：规则后端/模拟后端可以根据通知、应用切换、屏幕状态、系统状态等信号生成候选 intent；
- **策略校验**：PolicyEngine 根据风险、置信度和动作类型过滤不安全建议；
- **动作执行骨架**：DefaultActionExecutor 和 OfflineAdapter 提供动作执行与模拟接口；
- **动作总线**：ActionBus 使用 channel 解耦采集、处理和执行链路。

7.4 测试与验证

中期阶段已经建立了覆盖核心模块的测试体系，重点包括：

- 隐私脱敏：验证 raw PII 不进入 SanitizedEvent 或 StructuredContext；
- 窗口聚合：验证 foreground apps、notified apps、semantic hints、system status summary 的构建；
- 策略引擎：验证风险等级、置信度、动作白名单和 target 检查；
- 动作执行：验证不同 action type 的执行骨架和错误处理；
- 模拟决策：验证多类信号能被转为 intent/action；
- 动作总线：验证采集与处理之间的异步边界。

GoldenTrace 数据结构和离线 replay 方向已经确定。中期之后，项目需要把它从测试骨架推进为稳定的回归基线。

8 研究背景与问题动机

8.1 移动端体验痛点

移动端用户经常在聊天、浏览器、文档、开发工具或办公应用之间切换。传统系统通常在资源压力出现后才被动回收或重建进程；当用户再次打开应用时，Dex 加载、资源解析、数据库初始化和后台状态恢复都可能造成可感知延迟。

DiPECS 的思路是把部分冷启动成本提前：在高置信度场景中，系统可以保持短窗口内的关键进程或缓存资源，将部分冷启动转为热启动。中期阶段的目标不是立刻证明所有场景都有收益，而是先建立能够安全提出和审查这类动作的系统框架。FALCON 等移动端预启动工作说明，使用位置、时间等上下文预测 app 启动并做 prelaunch 是可研究的系统优化方向[7]。

8.2 语义鸿沟

Android 能提供大量低层事件，例如包名、时间戳、UID、PID、通知 channel、网络和内存状态，但系统并不知道用户为什么切换应用、某条通知是否重要、某个进程是

否值得保留。换言之，系统知道 What/When，却缺少 Why 和 What-Inside。

DiPECS 尝试在 PID 级事件和 intent 级语义之间建立中间层：用脱敏上下文和规则/模型生成候选意图，再用本地策略决定动作。这是项目相对普通日志采集器的主要区别。

9 技术路线

9.1 Android 信息来源

中期设计把信息来源分为三层，见表 5。

表 5: 中期规划中的 Android 信息来源

层级	代表来源
公开 API	UsageStats、NotificationListener、Battery、Network、Screen state。
系统特权/开发者接口	Accessibility、ADB、dumpsys、Shizuku 或 root 环境。
内核事件	eBPF、Binder trace、文件 I/O、进程生命周期和内存压力事件。

中期原型主要落在第一层，确保普通授权环境下可运行；第二、三层作为后续 AOSP/root 环境验证方向。

9.2 多源信息融合

多源信息融合的核心是统一 schema。不同采集源输出格式不同、粒度不同、隐私风险不同，必须先归一化为 RawEvent，再通过 PrivacyAirGap 进入 SanitizedEvent。窗口聚合之后，DecisionRouter 和 PolicyEngine 只面对统一的 StructuredContext，不需要理解每个采集源的原始细节。

这种设计使系统可以逐步扩展数据源。新增采集源只要能映射到 RawEvent 或 CollectorEnvelope，就能复用脱敏、聚合、决策和审计链路。

9.3 资源调度策略

中期阶段定义了三类低风险动作方向，见表 6。

表 6: 中期阶段定义的动作方向

动作方向	触发条件	内容
KeepAlive	高置信度、低风险	短窗口保护相关进程或本系统任务，减少重建成本。
PreWarm / Prefetch	中高置信度、target 明确	预读可访问资源或预热本应用/安全资源。
Hint/NoOp	低置信度或风险不明	只记录候选，不做实际资源占用。

实际执行时，动作必须经过 policy 检查。中期阶段更关注动作生命周期和策略边界，而不是直接追求强系统控制能力。

10 当前局限

中期阶段仍存在以下局限：

(1) Android 到 Rust 的生产入口仍需进一步稳定，JNI、JSONL 或 daemon 入口需要明确主路径；

(2) 云端 LLM 当时仍处于 mock/proxy 或接口设计阶段，真实云端后端需要后续接入；

(3) GoldenTrace 和 replay 还需要扩充数据集，覆盖通勤、学习、社交、办公等典型场景；

(4) 真实 Android action bridge 尚不完整，许多动作只能离线模拟或本地 stub；

(5) 内核级信息抽取、eBPF、Binder trace 和文件热点画像还停留在后续 root/AOSP 阶段。

11 后续计划

中期之后的工作计划包括：

(1) 完成 Android 原型端到端演示：采集、脱敏、预测、策略校验和本地动作记录；

(2) 建立稳定 Golden Trace 数据集，并将 replay/audit hash 纳入回归测试；

(3) 对比无预热、启发式规则和模型预测三种策略的动作数量、拒绝率、延迟收益和资源开销；

(4) 接入真实云端 LLM 后端，同时保证网络失败时可以回退到 RuleBased 或 NoOp；

(5) 在权限更高的环境中验证 eBPF、Binder 和文件预读等增强能力，但不把它们作为普通 APK 阶段的前提。

12 结论

中期阶段，DiPECS 已经完成从概念到最小可执行原型的主要跨越。项目证明了在 Android 公开 API 和 Rust 核心链路基础上，可以建立一条可脱敏、可聚合、可决策、可审查、可回放的数据流。接下来的关键不再是单纯增加更多规则，而是把 Android 真实输入、云端后端、golden trace 和 Android-safe 动作桥接接到同一条链路上，并用实验指标验证预测与优化是否真正带来收益。

参考文献

[1] Android Developers. UsageStatsManager. Android Developers, 2024.

[2] Android Developers. NotificationListenerService. Android Developers, 2024.

[3] Android Open Source Project. Extend the kernel with eBPF. AOSP, 2024.

[4] Android Developers. BatteryManager. Android Developers, 2024.

[5] Android Developers. PowerManager. Android Developers, 2024.

[6] Android Developers. AccessibilityService. Android Developers, 2024.

[7] Tingxin Yan, David Chu, Deepak Ganesan, et al. FALCON: Fast app launching via predictive user context. Microsoft Research, 2012.