DiPECS 相关工作调研报告

——端侧 AIOS 的数据边界、决策边界与动作边界

调研报告

项目名称：DiPECS

作者单位：DiPECS Team

报告类型：调研报告

完成时间：2026 年 3 月 28 日

目录

1 引言 2

2 研究背景 3

3 调研范围与方法 3

4 项目定位与技术特性 4

5 AIOS 与操作系统级智能代理 5

6 移动应用行为预测 5

7 隐私保护与上下文建模 6

8 Android 数据源能力 6

9 eBPF 与内核观测 7

10 动作治理与可审计执行 8

11 相关系统对比 8

12 研究局限与展望 9

13 结论 10

DiPECS 相关工作调研报告

——端侧 AIOS 的数据边界、决策边界与动作边界

摘要： **目的：**明确 DiPECS 端侧 AIOS 原型在数据采集、隐私脱敏、上下文聚合、决策路由与动作治理各层之间应建立何种边界与关系。**方法：**以公开学术论文、Android 官方文档、Google Play 政策以及项目代码仓库为资料来源，调研 AIOS、移动应用行为预测、隐私保护上下文建模、Android 数据源能力与动作治理等方向的研究现状。**主要发现：**现有工作分别验证了系统级代理、跨应用自动化和移动上下文预测的可行性，但在 Android/Linux 端侧场景中，权限边界、隐私边界和动作边界仍是未充分解决的问题。**结论：**DiPECS 的技术定位可概括为：以 Android 公开 API 和 Linux 本机信号为默认数据源，以本地规则为决策基线，以隐私空气隙和策略审查作为核心约束的受控端侧 AIOS 原型；后续工作应在不破坏上述边界的前提下，逐步扩展数据源、接入云端模型并完善可审计的动作执行链路。

关键词： 端侧 AIOS；移动行为预测；隐私脱敏；动作治理；Android 数据源

1 引言

DiPECS 是一个面向 Android/Linux 端侧环境的意图预测与动作治理原型系统。与通用聊天助手或完全自治的桌面代理不同，DiPECS 的研究问题可概括为：在现有操作系统权限边界内，如何把本地可观测信号整理成结构化上下文，再由本地规则或云端模型产生候选意图，最后通过策略引擎和动作生命周期模块决定哪些低风险动作可以执行。

从工程实现看，DiPECS 当前已经形成一条可回放的数据链路：Android JSONL / 守护进程源 / 离线测试夹具产生 `RawEvent`，经 `PrivacyAirGap` 转换为 `SanitizedEvent`，再由 `WindowAggregator` 构造 `StructuredContext`，随后由 `DecisionRouter` 选择后端生成意图，经 `PolicyEngine` 审查后形成 `AuthorizedAction`，最终输出 `AuditRecord` 用于回放与追责。该链路可用图 1 概括。

Android JSONL / daemon sources / replay fixture → `RawEvent` → `PrivacyAirGap` → `SanitizedEvent` → `StructuredContext` → `DecisionRouter` → `PolicyEngine` → `AuthorizedAction` → `AuditRecord`

图 1: DiPECS 数据链路概览

上述链路决定了本调研的重点。本文不把 DiPECS 定位为通用聊天助手，也不把它定位为单一的 next-app prediction 模型，而是把它看作一个端侧 AIOS 原型。它需要同

时回答三个问题：哪些上下文可以被安全采集，哪些意图可以被本地或云端判断，哪些动作可以在策略约束下落地。

2 研究背景

大语言模型推动了智能代理系统的发展。现有研究表明，模型可以在代码仓库、桌面应用、网页和多轮任务中承担计划、推理和执行职责[1]。然而，当智能代理进入操作系统或移动终端场景时，问题不再只是“模型能不能完成任务”，还包括以下系统问题：

(1) 终端侧数据来自多个异构来源，包括应用前后台切换、通知、屏幕状态、系统负载和进程变化；

(2) 原始数据常包含个人信息，不能直接交给模型或外部服务；

(3) 模型输出不能直接变成系统动作，必须经过权限、风险和上下文一致性检查；

(4) 系统需要可回放、可审计，否则难以验证行为是否符合策略。

因此，端侧 AIOS 的关键不只是“预测用户下一步会做什么”，而是要在预测之前建立数据边界，在预测之后建立动作边界。这个背景也解释了为什么 DiPECS 当前实现中，Rust core、replay、policy 和 audit 占据了很大比例：这些模块决定系统是否可验证、可降级和可追责。

3 调研范围与方法

参考操作系统级智能代理（OSH）领域的调研结构，本报告按“系统背景—关键技术—相关系统—项目差异”的顺序组织。具体调研范围包括：

- AIOS 与操作系统级智能代理：说明 LLM agent 如何被系统化，以及这些工作与传统 OS 边界的关系；
- 移动应用行为预测：说明应用切换、通知、时间和设备状态是否足以支持下一步意图推断；
- 隐私保护上下文建模：说明为什么不能把原始事件直接交给模型，以及本地脱敏的必要性；

- Android 数据源与系统观测：说明公开 API、Accessibility、Binder、eBPF 等数据源的能力和限制；
- 动作治理与可审计执行：说明预测结果为什么还需要 policy、capability 和 audit 约束。

资料来源以公开发表的学术论文、Android 官方文档、Google Play 政策和项目代码仓库为主，检索截至 2026 年 3 月 28 日。

4 项目定位与技术特性

与许多以模型能力为中心的 Agent 项目相比，DiPECS 的主要特点在于系统边界更明确。表 1 对其核心技术特性进行了归纳。

表 1: DiPECS 核心技术特性

特性	说明
事件驱动	系统从 Android 事件、Linux 进程状态和离线 trace 出发，而不是从自然语言请求出发。
隐私前置	通知正文、文件路径、notification key 等敏感信息在进入决策后端前被转换为摘要或直接丢弃。
后端可替换	`DecisionRouter` 可以选择 RuleBased、CloudLlm 或 FallbackNoOp，后续也可接入统计预测模型。
动作受控	模型或规则只能提出 `IntentBatch`；动作能否执行由 capability、policy 和 lifecycle 决定。
可回放	同一 JSONL trace 可以离线重跑，用于比较 intent、policy verdict、audit hash 和 denial reason。

这些特性共同构成 DiPECS 的受控端侧 AIOS 定位：强调边界先于能力，可审计先于自动化。

5 AIOS 与操作系统级智能代理

AIOS 相关工作把大语言模型代理从单个应用提升到系统级平台。典型观点认为，LLM 可以承担类似操作系统服务的角色，为多个代理提供调度、上下文、记忆、工具调用和权限管理能力；AIOS 实现强调将代理应用逻辑和底层模型服务分离，使多个代理能够共享模型资源和系统能力[2;3]。

OS-Copilot、FRIDAY、UFO 等工作则从桌面或通用计算机代理出发，让模型通过终端、文件系统、图形界面或应用 API 完成复杂任务。这些系统证明了“代理作为系统用户”的可行性，但它们通常面向桌面环境，依赖窗口自动化、视觉理解或高权限工具接口。移动端 Android 场景存在更强的权限边界：普通应用无法任意读取其他应用内部状态，也不能随意注入系统级动作[4–6]。

DiPECS 与这些工作的关系是继承其系统化思想，但收窄执行边界。当前实现没有把云端 LLM 作为默认内核，也没有默认授予代理高权限自动化能力。相反，系统把代理输出限制为意图批次，再由策略引擎和动作生命周期模块决定是否形成授权动作。这使得 DiPECS 更接近一个“受控端侧 AIOS 原型”，而不是完全自治的桌面代理。

6 移动应用行为预测

移动场景中的行为预测研究早于 LLM 代理。FALCON 等系统使用时间、地点、传感器和历史使用模式预测用户下一次启动的应用，以降低启动延迟和能耗。AppUsage2Vec、DeepAPP、Appformer 以及后续基于 Transformer 或图神经网络的方法进一步把应用序列、时间上下文和用户个性化编码进预测模型，用于 next-app prediction 或应用推荐[7–10]。

这些方法为 DiPECS 提供了两点启发。第一，移动端上下文确实包含可预测的规律，应用切换、通知到达、屏幕状态和系统状态都可以构成意图推断信号。第二，仅有预测结果不足以支撑系统动作。传统 next-app prediction 主要输出“用户可能打开什么应用”，而 DiPECS 需要处理的是“系统是否应该提出某个动作、该动作是否安全、是否需要授权以及如何留下审计记录”。

因此，DiPECS 当前没有把核心问题定义为最高精度的应用预测，而是定义为从多源事件生成结构化上下文，再在策略约束下产生可审计动作建议。预测模型可以作为未来后端之一接入 `DecisionRouter`，但不应绕过隐私和动作治理边界。

7 隐私保护与上下文建模

端侧智能系统的关键风险是原始上下文泄漏。通知标题、正文、文件路径、应用包名、交互时间等数据都可能间接暴露用户身份、工作内容或社交关系。相关研究通常采用本地处理、最小化采集、脱敏摘要、访问控制或联邦学习等方式降低风险。

DiPECS 当前实现采用明确的隐私空气隙设计：`RawEvent` 只允许存在于采集器到 core 的短路径内，经过 `PrivacyAirGap` 后，后续模块只看到 `SanitizedEvent` 和 `StructuredContext`。例如，通知文本不会直接进入决策后端，而是转换为文本长度、书写系统和语义提示；文件路径只用于类型分类，不保留原始路径；Android notification key 等可能携带个人信息的字段会被丢弃。

这一设计和许多 LLM 代理系统不同：后者常把完整屏幕、网页或文件内容交给模型，以换取更强理解能力。DiPECS 选择牺牲一部分语义完整性，换取更清晰的隐私边界和可验证的数据流。对于 Android/Linux 端侧系统，这种取舍更符合普通用户授权和最小权限原则。

8 Android 数据源能力

Android 平台的数据源天然分层。普通应用可以通过用户授权访问部分公开 API，例如 UsageStats、NotificationListener、BatteryManager、ConnectivityManager 和 PowerManager。这些 API 能提供应用前后台切换、通知元数据、屏幕交互、电量和网络状态等信息，是 DiPECS 当前 Android collector 的主要基础[11–13]。

更高权限的数据源包括 `AccessibilityService`、ADB/Shizuku、Root、Xposed 或内核级追踪。它们可以提高观测精度，但代价是权限门槛、部署成本、应用商店合规性和隐私风险显著增加。DiPECS 当前把无障碍服务定位为筛选性来源：可以用于界面预览和调研，但如果事件没有符合 Rust schema 的 `rawEvent`，Rust 入口会跳过该行[14;15]。表 2 对数据源分层进行了总结。

表 2: Android 数据源分层与项目定位

数据源层级	代表机制	对 DiPECS 的意义
公开 API	UsageStats、NotificationListener、设备状态	当前真实 Android 采集主路径，适合默认启用。
无障碍服务	AccessibilityService	可作为增强观测来源，但当前不是主链路 schema。
开发者接口	ADB、Shizuku、dumpsys	适合实验，不适合普通用户默认部署。
Root/内核级	eBPF、Xposed、LSPosed	部署和合规成本高，不属于当前默认目标。

9 eBPF 与内核观测

eBPF 在 Linux 系统观测、网络过滤和安全分析中非常重要。它可以在内核侧以较低开销采集系统调用、网络包、调度事件或文件访问等信号。然而，Android 应用意图往往存在于应用层语义中，例如通知文本、聊天 UI、文件消息类型或用户点击含义；仅靠内核事件很难还原这些语义[16]。

从当前项目目标看，eBPF 不是解决 Android 语义采集的主路径。即使能够观察到某个进程调用 `open()`、`write()` 或网络发送，也不能可靠判断 “用户是否收到一个工作文件” 或 “是否应该建议打开某个文档”。此外，Android 上部署 eBPF 还会受到 root 权限、SELinux、内核版本和系统策略限制。

仓库中已经预留了 `BinderProbe` 相关接口，但当前实现只是检测桩，不产生真实 Binder 事件。因此报告和后续设计都应把 eBPF 描述为 “未来可研究的低层观测方向”，而不是当前已经接入的生产数据源。

10 动作治理与可审计执行

智能代理系统中的动作执行通常比意图识别具有更高的安全风险。桌面代理可以通过 UI 自动化或 shell 命令直接操作文件、窗口和网络；移动端系统如果采用类似方式，则会遇到更强的权限和隐私约束。相关工作中的 guardrail、权限控制和人工确认机制表明，模型输出必须经过单独的安全边界[1]。

DiPECS 当前把这条边界拆成三层。第一层是后端能力等级：RuleBased 只能产生低风险的 `NoOp`、`KeepAlive` 和 `ReleaseMemory`，而 `PreWarmProcess`、`PrefetchFile` 等动作需要更高能力后端。第二层是 `PolicyEngine`：它检查风险等级、置信度、动作类型、紧急程度和 target 是否来自当前上下文。第三层是 `ActionLifecycle`：它把每个候选动作展开为 proposal，经过 schema validation、policy verdict、adapter dispatch，最终形成一条终态明确的 `AuditRecord`。

这使得 DiPECS 和“模型直接调用工具”的系统不同。即使云端模型返回了结构化动作，本地端也不会立即执行；所有动作都要经过可解释的拒绝理由或授权记录。因此，调研中与 DiPECS 最相关的不是单纯的工具调用框架，而是具备机制/策略分离、最小权限和可审计执行思想的系统设计。

11 相关系统对比

综合上述方向，可以把相关系统分为四类，表 3 给出了对比。

表 3: 相关方向与 DiPECS 当前定位

方向	主要贡献	可借鉴点	与 DiPECS 的差异
AIOS / LLM OS	把 LLM 服务、代理调度、记忆和工具调用系统化	把 Agent 能力放进系统层，而非单个应用脚本	多数工作更关注模型资源和代理框架，较少落到 Android 端侧权限边界。
桌面智能代理	通过 GUI、终端或应用 API 完成跨应用任务	说明跨应用任务需要工具边界和状态反馈	通常依赖桌面自动化能力，Android 普通应用环境没有同等权限。
移动应用预测	使用历史序列和上下文预测下一应用或下一行为	支持后续接入统计或序列预测后端	关注预测精度，较少处理动作授权、审计和隐私空气隙。
隐私保护上下文系统	本地处理、最小化采集、脱敏和访问控制	支撑 RawEvent 到 StructuredContext 的空气隙设计	脱敏会降低语义完整性，需要规则和模型适配。
Android 数据源	公开 API、Accessibility、Binder、eBPF 等分层观测	明确哪些数据源可默认启用，哪些只能做实验	高权限数据源部署成本高，不能作为当前主链路前提。
动作治理系统	capability、policy、audit、guardrail	支撑模型建议与真实执行分离	需要牺牲部分自动化能力，换取可解释拒绝和可审计终态。

12 研究局限与展望

本调研存在以下局限：

(1) 以公开文献和官方文档为主，未进行大规模的独立用户实验或真实设备评测；

(2) 对高权限数据源（Accessibility、Binder、eBPF）的讨论以能力分析为主，未给出

详细的部署方案；

(3) 对隐私空气隙的评估主要基于项目内部测试，缺少第三方审计。

后续工作可在不破坏隐私、策略和动作边界的前提下继续推进：扩展 Android raw event schema、完善 Android 动作桥接、引入可替换的预测模型或 LLM 后端，并把 eBPF/Binder 观测作为独立实验方向评估，而不是作为当前主链路的前提。

13 结论

相关工作显示，AIOS、桌面代理和移动行为预测已经分别证明了系统级代理、跨应用自动化和移动上下文预测的可行性。但在 Android/Linux 端侧环境中，真正困难的是权限边界、隐私边界和动作边界。DiPECS 当前实现选择了一条保守但可验证的路线：用公开 API 和本机信号构造结构化上下文，用隐私空气隙限制模型可见信息，用策略引擎限制动作输出，并用 replay 和审计哈希验证行为。

本文的调研结论对当前代码路线有直接约束：RuleBased 后端应保持为低权限本地基线，云端 LLM 应作为可选增强，Android 采集源应按权限层级逐步推进，下一阶段更重要的工作是补齐可评价指标（policy denial、noop rate、audit hash 稳定性、动作执行终态）。

参考文献

[1] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey. arXiv:2309.07864, 2023.

[2] Kai Mei, Shiyao Zhang, Jiawei Liu, et al. Aios: LLM agent operating system. arXiv:2403.16971, 2024.

[3] Charles Packer, Vivian Fang, Shishir Patil, et al. MemGPT: Towards LLMs as operating systems. arXiv:2310.08560, 2023.

[4] Zhiyong Wu, Chengcheng Han, Zichen Ding, et al. OS-Copilot: Towards generalist computer agents with self-improvement. arXiv:2402.07456, 2024.

[5] OS-Copilot project page. Project Page, 2024.

[6] Chaoyun Zhang, Lichao Sun, Yingjie Li, et al. UFO: A UI-focused agent for Windows OS interaction. arXiv:2402.07939, 2024.

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

[8] Sha Zhao, Julian Ramos, Ziwen Jiang, et al. AppUsage2Vec: Modeling smartphone app usage for prediction. IEEE Xplore, 2019.

[9] Zhihao Shen, Zhefu Yan, Zhongmin Chen, et al. DeepAPP: A deep reinforcement learning framework for mobile application usage prediction. ACM Digital Library, 2019.

[10] Tong Li, Tong Xu, Huangdong Wang, Zhen Tu, Sasu Tarkoma, Zhu Han, and Pan Hui. Smartphone app usage analysis: Datasets, methods, and applications. *IEEE Communications Surveys & Tutorials*, 24(2):937–966, 2022.

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

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

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

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

[15] Google Play Console Help. AccessibilityService API policy. Google Play Console Help, 2024.

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