Mobile OS Memory Management System

Project Report (Direct PDF Generation)

This PDF is generated directly from Python code and project understanding, not from Markdown conversion.

1. Project Goal

The project builds a real-time Android memory monitoring and optimisation dashboard using ADB, Python, and Streamlit. It provides visibility into memory consumption and allows safe user-driven cleanup of low-priority apps.

2. Core Workflow

- Detect whether an Android device is connected through ADB.
- Collect system memory from dumpsys meminfo (with /proc/meminfo fallback).
- Collect per-process PSS and OOM priority from dumpsys outputs.
- Score processes by kill safety and estimate freeable memory.
- Render results in a Streamlit dashboard with charts and action buttons.

3. Main Modules

- config.py: thresholds, OOM priority mapping, kill blocklist, refresh settings.
- modules/adb_utils.py: ADB discovery, shell execution, connectivity checks, force-stop command.
- modules/memory_reader.py: parses total/used/free/lost RAM.
- modules/process_reader.py: extracts process PSS and OOM codes, supports Android 15 format.
- modules/smart_manager.py: recommendation logic and kill-candidate filtering.
- modules/demo_data.py: realistic simulated data when no device is available.
- app.py: full dashboard UI with 5 tabs and glassmorphism theme.

4. Decision Logic

- Usage >= 80%: Critical memory pressure.
- Usage >= 60%: Warning state.
- Usage < 60%: Healthy state.
- Candidate process must have kill_score >= 3 and must not be in blocklist.
- Candidates are sorted by highest PSS first for maximum memory recovery.

5. Dashboard Features

- Memory Overview: metric cards, gauge, progress bar, status alerts.
- Running Processes: table, top consumers chart, per-app stop buttons.
- History: time-series graphs for used/free memory and usage percent.
- Smart Recommendations: kill candidates, estimated freeable MB, pie chart, optimize-all.

- Android vs Our Model: comparison table for academic presentation.

6. Reliability and Safety

- Automatic demo fallback when device/ADB is unavailable.
- System-critical package blocklist prevents unsafe force-stop actions.
- Cache TTL and optional 30s auto-refresh reduce command load and lag.
- ADB path auto-discovery supports common Windows installation paths.

7. Technology Stack

- Python 3.14
- Streamlit 1.53
- Plotly 6.5
- Pandas 2.3
- Android Debug Bridge (ADB)

8. Conclusion

This project successfully demonstrates an adaptive memory management interface over Android's low-level process state. It improves user visibility, control, and practical understanding of OS memory behavior while remaining safe through priority-based filtering and blocklist protection.