

Resuscitation

v2.2.0.125

Péter Felecan

assisted by Claude Sonnet 5 (Anthropic)

September 5, 2026

Contents

1	Introduction	3
2	STARTED Google Nexus 5 (lg-hammerhead)	3
2.1	Target	4
2.2	Prerequisites	4
2.3	Procedure (confirmed working)	4
2.4	Post-boot: internet access and package management	6
2.4.1	Getting SSH working at all (prerequisite, easy to miss)	6
2.4.2	Internet access via the host	6
2.4.3	Privilege escalation: sudo, not doas	7
2.4.4	Package management (Alpine's apk)	7
2.4.5	WiFi AP mode: confirmed supported	7
2.5	Sensor inventory (confirmed against the actual unit)	7
2.6	Power management (confirmed limitation)	8
2.7	Sensor telemetry: dudhya-sensord (MQTT publisher)	9
2.7.1	Topic tree published	9
2.7.2	Identification and liveness: the status topic	10
2.7.3	Getting the source onto the device	10
2.7.4	Build prerequisites (on-device)	10
2.7.5	Runtime prerequisites (on-device)	11
2.7.6	Build, configure, and run	11
2.7.7	Sensor corrections	12
2.7.8	Open items	12
2.8	WiFi: station/AP mode switching	13
2.8.1	Prerequisites (on-device)	13
2.8.2	Configuration	13
2.8.3	Switching	14

2.8.4	Open items	14
2.9	Provisioning: scripted package installation/upgrade	15
2.9.1	On the host: <code>dudhya-host-connect</code>	15
2.9.2	On the device: <code>dudhya-device-update</code>	16
2.9.3	Is cable and radio mutually exclusive?	16
2.9.4	Known quirk: USB not detected if the cable is connected after boot, not before	16
2.9.5	Known quirk: host reboot can assign the wired ethernet and USB gadget the same interface name	17
2.9.6	Open items	17
2.10	Precedent	18
3	TODO Google Pixel 3a (<code>google-sargo</code>)	18
3.1	Target	18
3.2	Prerequisites	18
3.3	Procedure (planned, not yet attempted)	19
3.4	Known caveats (from prior research — expect these, don't discover them blind)	19
3.5	Sensor inventory (not yet tested)	19
3.6	Power management: running without a battery	20
3.6.1	Complete wiring specification (matches the now-verified schematic file)	22
4	Annex: Second-device selection research	23
4.1	Summary and recommendation	24
4.2	Ranking table	24
4.3	Per-device findings	25
4.3.1	Xiaomi Poco F1 (<code>xiaomi-beryllium</code>) — top pick	25
4.3.2	OnePlus 6 (<code>oneplus-enchilada</code>) — strong alternate	25
4.3.3	Google Pixel 3a (<code>google-sargo</code>) — alternate with a GPS caveat	26
4.3.4	OnePlus 6T (<code>oneplus-fajita</code>)	26
4.3.5	Fairphone 4 (<code>fairphone-fp4</code>) — not recommended	26
4.3.6	Samsung Galaxy S9 (<code>samsung-starlte</code>) — not recommended	26
4.3.7	Nokia N900 (<code>nokia-n900</code>) — GPS confirmed, hardware too weak	27
4.4	Physical dimensions and weight	27
4.5	Battery-free (external DC) operation feasibility	27
4.6	Cybersecurity and binary-blob posture	28
4.7	Purpose-built mainline phones (secondary by project ethos)	30
4.8	Recommended validation sequence, once a unit is acquired	30
4.9	Caveats	30

5	Annex: Operating system choice re-evaluation	31
5.1	Summary and recommendation	31
5.2	Candidate classification	32
5.2.1	Genuine mainline-Linux-kernel-first (primary tier)	32
5.2.2	Vendor-kernel/Halium-shim (structurally lower tier, not excluded outright)	33
5.2.3	Excluded (Android-based or proprietary)	33
5.3	Criterion: genuine mainline-kernel model	33
5.4	Criterion: community strength	33
5.5	Criterion: device support breadth and depth	34
5.6	Criterion: sensor set and GPS completeness	35
5.7	Criterion: project health and activity	35
5.8	Criterion: developer ergonomics	36
5.9	Recommendations	36
5.10	Caveats	36

List of Figures

List of Tables

1 Introduction

This document captures the resuscitation procedures for smartphones (per [Concept.pdf](#) and the scoping decision recorded in [ResearchJournal.pdf](#)).

2 **STARTED** Google Nexus 5 (lg-hammerhead)

Status: **confirmed working end-to-end** against a real unit, on a fresh system: unlock, `pmbootstrap init` with OpenRC (not systemd — see step 4), install, flashing, and first boot all succeed; the device reaches a getty login prompt on its own screen, brings up USB networking (NCM, registering as `enx<mac>` on the host), and is reachable over SSH, showing the standard postmarketOS MOTD on login. Using OpenRC rather than systemd (see step 4) resolves a "stuck at logo, no network" issue systemd otherwise causes on this device (see ResearchJournal.org for the full account, including a false alarm along the way: `lsusb` mislabels the device "Nexus 4 (fastboot)" purely because of a static ID-database lookup keyed on a reused vendor:product pair — `dmesg` shows the real, correct identity and confirms the USB gadget came up properly; nothing was actually wrong there).

If SSH doesn't come up immediately after flashing: the host may need a DHCP client bound to the new USB network interface manually, e.g. `dhclient enx<mac>` (interface name varies by host), before `ssh <user>@172.16.42.1` works.

Next phase: verify sensor/telemetry access (GPS, accelerometer, gyroscope — see the open FutureResearch.org item), which is the project's actual goal for this device, not just getting it to boot.

2.1 Target

- Device codename: `hammerhead`, vendor `lg`.
- Target OS: postmarketOS, edge channel (see the scope decision's OS-choice rationale).
- Goal for this device: a headless/telemetry node, not a phone UI — so a minimal or no graphical user interface is preferred over a full desktop session.

2.2 Prerequisites

- A Linux host machine (physical or VM) to run `pmbootstrap`; native Windows and WSL are both reported as troublesome for this step, a plain Linux install is the reliable option.
- A USB cable and `adb~/~fastboot` on the host.
- The device fully backed up: unlocking the bootloader wipes all data.

2.3 Procedure (confirmed working)

1. On the device, enable Developer Options, then enable "USB debugging". Look for "OEM unlocking" too, but **do not treat its absence as a blocker**: on a real unit (Android 6.0.1, checked in both French and English) this toggle does not appear at all. This matches other Nexus 5 owners' reports for Android 6 — the device predates the newer bootloader-security model that hard-gates `fastboot` behind that toggle on later devices (5X, 6P); on the Nexus 5's older bootloader either the toggle doesn't apply or doesn't render, but it does not block the unlock command below.
2. Reboot to the bootloader (power off, then hold Volume Down and press Power), connect the USB cable, and on the host run: `fastboot oem unlock` — the **old**-style command; the Nexus 5 does not use the newer `fastboot flashing unlock` syntax some later devices require. `fastboot`'s own help text may not even list `oem unlock` as recognised; run it anyway. Confirm the prompt on the device itself using the volume buttons to choose and the power button to select.

3. Behaviour after confirming the unlock varies by unit: the postmarketOS wiki describes the device returning automatically to the Fastboot Mode screen, but a real unit has instead rebooted into Android's setup wizard (userdata was wiped, so Android treats it as factory-fresh, which resets Developer Options and its toggles with it). Either outcome is fine and needs no special handling: the bootloader-unlock state is a firmware-level flag that persists independently of anything on the Android/userdata side, and every remaining step below is a plain **fastboot** command — fastboot mode does not depend on Developer Options or USB debugging at all (that setting only gates **adb** access from a **running** Android OS). There is no need to complete the setup wizard or re-enable Developer Options/USB debugging: just power off and re-enter the bootloader (Volume Down + Power) and continue directly with the next step.
4. On the host, install **pmbootstrap** and run **pmbootstrap init**, selecting vendor **lg**, device codename **hammerhead**, the **edge** channel, and the **console** UI (headless, no graphical environment) rather than a full desktop, per the telemetry-node goal above. When asked about using **systemd** as the init system: answer **never** (i.e. use OpenRC). **systemd** stalls at first boot on this device; every actual working reference found for this device used OpenRC, and postmarketOS's own current status shows **systemd** support as recent (merged into edge January 2025) with an acknowledged, still-being-reworked USB-gadget-handling story on mainline devices — see ResearchJournal.org for the full reasoning. **Not yet re-verified with OpenRC**; correct this note again once it is.
5. Run **pmbootstrap install** with no flag (current **pmbootstrap** has no **--no-fde** flag: no full-disk encryption is now the default behaviour, flipped from an earlier opt-out model to an opt-in one — add **--fde** only if full-disk encryption is actually wanted). This is the step that builds the rootfs/ kernel images in a local chroot on the host — it downloads packages, and interactively prompts to set a login password for the account named during **init**. Nothing exists to flash before this step runs.
6. With the device back in fastboot mode, run the flasher commands **in the order pmbootstrap install itself prints** at the end of its own output ("FLASHING INFORMATION" block) for the current build. On the version used here, that order is: **pmbootstrap flasher flash_lk2nd** first (pmbootstrap's own message: "Your device supports and may even require flashing lk2nd. You should flash it before flashing anything else."), then **pmbootstrap flasher flash_rootfs**, then **pmbootstrap flasher flash_kernel**. This resolves the earlier discrepancy noted in ResearchJournal.org: an older install transcript for this device didn't surface the lk2nd message at all, but the current tool version does, confirming the postmarketOS wiki's original claim. lk2nd boots the kernel from the rootfs's own **/boot**, not from a separately-flashed boot partition — do not flash the boot partition directly.
7. Reboot. SSH is enabled by default: once booted, log in as the username chosen dur-

ing `pmbootstrap init` with the password set during `install`, over the USB network interface `pmOS` brings up (`ssh <user>@172.16.42.1` is the standard address for this).

Device-specific quirk to check during step 4: `hammerhead` shipped in two variants with different DDR vendors (Elpida and another); if flashing fails with an "Unknown chunk type" error, the bootloader itself likely needs updating first to a current factory image before retrying.

2.4 Post-boot: internet access and package management

Confirmed against a real unit, on the fresh system used to reach the milestone above.

2.4.1 Getting SSH working at all (prerequisite, easy to miss)

Confirmed across multiple sessions/systems that the host's USB network interface often comes up (`dmesg` shows `cdc_ncm` registering, `ip a` shows it **UP**) without ever getting an IP address automatically — SSH will not connect until it has one. This is a **separate** problem from internet access below, and comes first. The interface name itself (`enx<mac>`) is generated from the phone's own USB gadget MAC address, which is **not fixed across the phone's own reboots** — confirmed directly: the same host showed `enx8e26206b800e` in one session and `enx82a34ae872cd` in a later one, with nothing on the host side having changed in between, only the phone having rebooted. Always re-check `ip a` after **every** phone boot; don't assume a name from an earlier session, even on the same host.

```
ip a                                # find the interface (e.g. enx<mac>);
                                   # no "inet" line means no IP yet
sudo dhclient enx<mac>             # try this first
```

If `dhclient` isn't available or doesn't get an address, assign one directly instead (the interface is already up, so nothing else needs setting):

```
sudo ip addr add 172.16.42.2/24 dev enx<mac>
```

Then `ssh <user>@172.16.42.1` should work. Only once this is working does the internet-access step below become relevant.

2.4.2 Internet access via the host

The USB link is a point-to-point connection to the host only; reaching the internet (needed for `apk` below) requires the host to route and NAT for the phone.

- On the host (interface name varies by machine — check with `ip a` which one is the actual internet-facing NIC; it was `enp0s31f6` on this system):

```
sudo sysctl -w net.ipv4.ip_forward=1
sudo iptables -t nat -A POSTROUTING -o enp0s31f6 -j MASQUERADE
```

Neither of these persists across a host reboot (`sysctl -w` and `iptables -A` are both runtime-only changes) — confirmed directly: `apk` on the phone failed for every package with "DNS: transient error," which looked like a DNS problem but was actually this forwarding rule having quietly gone missing after the host restarted. If `apk` fails like this, check first before assuming DNS is actually broken:

```
cat /proc/sys/net/ipv4/ip_forward      # should read 1
sudo iptables -t nat -L POSTROUTING -n -v # should show a MASQUERADE line
```

and re-run the two commands above if either looks wrong.

- On the phone, over the existing SSH session (interface name on the phone's own side is usually `usb0`; check with `ip a` if not):

```
sudo ip route add default via 172.16.42.2 dev usb0
sudo sh -c 'echo nameserver 1.1.1.1 > /etc/resolv.conf'
```

2.4.3 Privilege escalation: `sudo`, not `doas`

postmarketOS switched new installs to `doas` by default a while back (a project-wide change), so `doas` was expected here. Checked rather than assumed: a real unit has no `doas` at all; `sudo` is present and works. Exact reason for the discrepancy not confirmed (older base image component? console UI specifically?) — noted as a real, checked finding rather than smoothed over. Use `sudo` for this device; re-check if a future fresh install shows otherwise.

2.4.4 Package management (Alpine's `apk`)

```
sudo apk update      # refresh the package index
sudo apk add <package> # install
sudo apk search <name> # search
```

2.4.5 WiFi AP mode: confirmed supported

`iw list` (after `sudo apk add iw`) lists AP under "Supported interface modes" on the actual unit — confirms the hardware and driver (`brcmfmac`, BCM4339) can run in access-point mode, which the chosen architecture (phone as the network hub, hosting its own MQTT broker) depends on. See the WiFi section below for the full, confirmed-working `hostapd` setup.

2.5 Sensor inventory (confirmed against the actual unit)

Full method and physical-test evidence in ResearchJournal.org's "Sensor inventory complete" entry; this is the reference summary for whoever builds the MQTT publisher next.

Working (all under `/sys/bus/iio/devices/` unless noted):

- `iio:device0` (apds9930) — ambient light (`in_illuminance0_input`) + proximity (`in_proximity0_raw`).
- `iio:device1` (mpu6515) — accelerometer (`in_accel_{x,y,z}_raw`) + gyroscope (`in_anglvel_{x,y,z}_raw`).
- `iio:device4` (bmp280) — barometric pressure (`in_pressure_raw/~_scale`) + temperature (`in_temp_raw/~_scale`).
- `iio:device5` (unnamed, AK8963-class) — magnetometer (`in_magn_{x,y,z}_raw`).
- `iio:device2~/~iio:device3` (PM8941 PMIC ADC channels) — exist, but are read-only power-management **measurement** channels (voltage/temperature-type telemetry for the PMIC's own internal use), not user sensors and not a power-control mechanism — ADC means Analog-to-Digital Converter, a reading-only capability, not an actuation path. Not telemetry candidates. (See FutureResearch.org's external-power item for the actual question this was raised alongside: a genuine software power-control path, if one exists on this PMIC, would be a different mechanism entirely and hasn't been looked for.)

Not working, with reasons (not just absence):

- **GPS/GNSS:** no `/dev/gnss*`, no `/dev/modem`, no `modem/QMI/rpmsg` activity in `dmesg` at all. Matches a known postmarketOS bug ("lg-hammerhead: modem not working" — the modem crashes during `ofono` initialization); GPS runs through this same modem's QMI Location Interface on Qualcomm platforms, so this is a real software-stack limitation for this device, not a separate GPS-chip fault.
- **Camera:** no `/dev/video*` at all. Not investigated further — Qualcomm camera ISPs are historically hard to mainline, an unimplemented camera on a community port is unsurprising, and it's outside Concept.org's actual goal (sensor telemetry, not imaging).
- On-device audio playback/recording: not implemented (documented upstream, not separately re-tested here); a Bluetooth audio device is the known workaround. Irrelevant to the telemetry goal.

2.6 Power management (confirmed limitation)

Full investigation and reasoning in ResearchJournal.org's corresponding entry; this is the reference summary.

Poweroff does not work — `poweroff`, `poweroff -f`, and `reboot` are all indistinguishable: every one of them just cycles the device back into the OS. Investigated rather than assumed: ruled out a watchdog-timeout bug (no watchdog driver active at all), ruled out a missing `lk2nd` boot-menu timeout (`/boot/extlinux/extlinux.conf` is already correctly configured), and found no restart/poweroff driver activity anywhere in `dmesg`. Cross-checked against masneyb/nexus-5-upstream's own exhaustive, patch-by-patch account of

every component mainlined for this device: it documents dozens of fixes down to minor bugs, and contains zero mentions of `poweroff/restart/~pshold~` anywhere. Conclusion: proper `poweroff` was most likely never implemented for this device's mainline port at all, not a bug to hunt down — a pattern also seen on several other community-mainlined phones (Redmi 2, Samsung Galaxy S3 mini, Nokia N900, Motorola Droid 4). **Not being pursued as a fix:** real kernel development work, outside this project's scope. The only reliable way to actually cut power is physical: Volume-Down+Power into the bootloader, then select "Power off" there.

Powering back on from a genuine cold poweroff needs "Start" selected manually in the bootloader — the power button alone does not resume it, even though `extlinux.conf`'s own `timeout~/ ~default` are correctly set. Plausible (not confirmed) explanation: this may be intentional `lk2nd` behaviour distinguishing a true cold power-on from a warm OS-triggered restart, rather than a bug.

Neither of these blocks testing `dudhya-sensord` — that only needs the phone booted and reachable, not powered off. Worth revisiting for an eventual vehicle deployment, where the better design may be surviving abrupt ignition-cut power (robust filesystem, maybe a small buffer capacitor) rather than relying on a graceful shutdown command at all.

2.7 Sensor telemetry: dudhya-sensord (MQTT publisher)

The phone hosts its own MQTT broker (see ResearchJournal.org for the design rationale) and publishes its confirmed-working sensors to it, as the first step toward being a self-contained network hub for in-vehicle deployment. Source: `dudhya-sensord.c`. Confirmed running on the physical unit itself (see "Open items" below).

2.7.1 Topic tree published

All topics are QoS 0, retained (so a late subscriber gets the last known value immediately rather than waiting for the next cycle):

Topic	Payload
<code>dudhya/<device>/accel</code>	<code>{"m_s2":{x,y,z},"g":{x,y,z},"ts"}</code>
<code>dudhya/<device>/gyro</code>	<code>{"rad_s":{x,y,z},"deg_s":{x,y,z},"ts"}</code>
<code>dudhya/<device>/magn</code>	<code>{"x","y","z","unit":"uT","ts"}</code>
<code>dudhya/<device>/pressure</code>	<code>{"value","unit":"hPa","ts"}</code>
<code>dudhya/<device>/temperature/board</code>	<code>{"value","unit":"C","ts"}</code> (chip die temp, not ambient)
<code>dudhya/<device>/light</code>	<code>{"value","unit":"lux","ts"}</code>
<code>dudhya/<device>/proximity</code>	<code>{"value","unit":"raw","ts"}</code> (uncalibrated, not a distance)
<code>dudhya/<device>/status</code>	<code>{"version","started_ts","ts","pid","sensors":{...}}</code>

`<device>` is `nexus5` by default (configurable — see below); the parent `dudhya/` prefix and per-device segment exist specifically so a future aggregator can subscribe to one sensor across every `dudhya` device (`dudhya/+/accel`) or to everything at once (`dudhya/#`), not just to one device in isolation.

2.7.2 Identification and liveness: the status topic

`dudhya/<device>/status` is published once at startup (retained, so any subscriber connecting later sees it immediately with no request needed) and again on demand: the daemon also subscribes to `dudhya/<device>/status/get`, and any message received there (content ignored) triggers an immediate re-publish with a fresh `ts`. `started_ts` stays fixed across refreshes, so `ts - started_ts` is the daemon's uptime; a changing `ts` on request is also a liveness check (confirms the daemon is still actually running right now, not just that it was at some point in the past — the retained message alone can't distinguish those two cases). Matches the request/response pattern already used elsewhere in this project's own infrastructure, layered on top of the passive retained publish rather than replacing it. Query it directly:

```
mosquitto_pub -h localhost -t 'dudhya/nexus5/status/get' -m ''
mosquitto_sub -h localhost -t 'dudhya/nexus5/status' -C 1
```

2.7.3 Getting the source onto the device

From the host, produce a distributable archive:

```
make dist
```

This produces `dudhya-VERSION.tar.gz` (automake's own dist mechanism — see README for why this differs from the project's own delivery archive convention: a different version-string suffix, otherwise the same file contents). Transfer it to the phone over the USB network link already established (see "Getting SSH working at all" above):

```
scp dudhya-VERSION.tar.gz <user>@172.16.42.1:~/
```

Then, on the phone itself over SSH, expand it and move into the resulting directory:

```
tar xzf dudhya-VERSION.tar.gz
cd dudhya-*/
```

From here, continue with the build steps below — run directly on the phone, per the native-build recommendation.

2.7.4 Build prerequisites (on-device)

This project's target (ARMv7 musl/Alpine, i.e. postmarketOS) differs from a typical x86_64 glibc development host in **both** architecture and libc. Recommended approach: build natively on the phone itself, reusing this project's existing `bootstrap && ./configure && make` workflow (already proven for the documentation build) rather than cross-compiling — this is one small C file, so the compile-time saving from cross-compiling would not be worth the real risk of a mismatched sysroot, which is a classic, hard-to-debug source of subtly wrong binaries.

```
sudo apk add build-base autoconf automake mosquitto-dev
```

`build-base` provides `gcc/make/musl-dev`; `autoconf~/~automake` are needed to run this project's own `./bootstrap`; `mosquitto-dev` provides the headers and `pkg-config` file `configure` checks for, and pulls in `mosquitto-libs` (the runtime shared library) as a dependency, covering the runtime need too — no separate library-only install is needed.

2.7.5 Runtime prerequisites (on-device)

```
sudo apk add mosquitto
```

This is the broker itself (a separate package from `mosquitto-dev~/~mosquitto-libs` above). Enable and start it before `dudhya-sensord`, since the daemon expects a broker already listening on `localhost:1883`:

```
sudo rc-update add mosquitto default
sudo rc-service mosquitto start
```

Confirmed and easy to miss (do this before testing with `mosquitto_sub~/~mosquitto_pub`): if the broker isn't actually started yet, `mosquitto_sub -h localhost -t 'dudhya/#' -v` fails with "Error: Bad file descriptor" — a confusing low-level message for "nothing is listening yet," not a real bug. Check `sudo rc-service mosquitto status` first if this happens; once the broker is running, `mosquitto_sub` connects and waits silently (interrupt with `Ctrl+C` — no output is the normal, correct behaviour with nothing published yet), and `mosquitto_pub -h localhost -t test -m hello` completes silently on success.

2.7.6 Build, configure, and run

```
./bootstrap
./configure
make
sudo mkdir -p /etc/dudhya
sudo cp sensord.conf.example /etc/dudhya/sensord.conf
# edit /etc/dudhya/sensord.conf if the defaults (device_name=nexus5,
# rate_hz=1.0, broker_host=localhost, broker_port=1883) need changing
```

Calibrate the accelerometer once, with the device lying flat, screen up and still (see the "Sensor corrections" section below for why this specific step needs a known orientation and cannot be automated the way the gyroscope's own zero-rate calibration is):

```
sudo ./dudhya-sensord --calibrate-accel /etc/dudhya/sensord.conf
```

Then install and start the service:

```
sudo cp dudhya-sensord /usr/local/bin/  
sudo cp dudhya-sensord.openrc /etc/init.d/dudhya-sensord  
sudo chmod +x /etc/init.d/dudhya-sensord  
sudo rc-update add dudhya-sensord default  
sudo rc-service dudhya-sensord start
```

Verify from another terminal (needs `mosquitto-clients`: `sudo apk add mosquitto-clients`):

```
mosquitto_sub -h localhost -t 'dudhya/#' -v
```

2.7.7 Sensor corrections

Applied after the first real on-device readings showed the accelerometer, gyroscope, and magnetometer all reading physically implausible values — see ResearchJournal.org for the full diagnosis (including tracing `in_accel*_calibbias` to the actual `inv_mpu6050` kernel driver source, confirming it is a live hardware register the chip applies before `raw` is ever read, not a separate value this daemon needs to add itself).

- **Magnetometer:** multiplied by a fixed x10 factor (`gMagnCorrectionFactor` in the source). Empirically derived from two real, differently-oriented on-device readings compared against Earth's expected field strength; not confirmed against the driver's own documented unit convention.
- **Gyroscope:** automatically zeroed at every startup (about a second of sampling, averaged, subtracted from every reading afterward). Safe to automate: a stationary gyro reads ~0 regardless of orientation.
- **Accelerometer:** needs the deliberate `--calibrate-accel` step above, since gravity's direction — unlike the gyroscope's zero-rate assumption — depends on orientation, which cannot be safely assumed automatically. Saves `accel_bias_x/y/z` into the config file; applied on every normal run afterward.
- **Not yet tested:** dynamic/moving scenarios (vehicle motion, active rotation) — verification so far is static, on a stationary device in two orientations; see FutureResearch.org.

2.7.8 Open items

- Confirmed on the physical unit: package install, the `mosquitto` broker running, both `mosquitto_sub~/ ~mosquitto_pub` connecting successfully, and `dudhya-sensord` itself built and running, publishing real telemetry.
- The OpenRC service script's `depend()` assumption that the `mosquitto` broker's own service is named exactly `mosquitto` is confirmed correct (`rc-service mosquitto status` works as named).

- The magnetometer's `uT` conversion needed an empirically-derived x10 correction once real data arrived — see "Sensor corrections" above. Not yet confirmed against the driver's own documented unit convention, and not yet tested under dynamic/ moving conditions (see FutureResearch.org).

2.8 WiFi: station/AP mode switching

Switches between station and access-point mode, not concurrent operation — concurrent use would need checking this chip/driver's actual supported interface **combinations** (a separate question from the plain mode-capability check already done via `iw list`), and switching covers the project's actual need without that extra verification step. **Not yet run against the physical unit** — see "Open items" below.

2.8.1 Prerequisites (on-device)

```
sudo apk add hostapd hostapd-openrc dnsmasq wpa_supplicant wpa_supplicant-openrc
```

`hostapd` and `~wpa_supplicant` each need their separate `-openrc` sub-package for the OpenRC service script; `dnsmasq` bundles its own. The DHCP client for station mode, `udhcpc`, is BusyBox- provided and needs no separate package.

2.8.2 Configuration

Three example config files ship with the project — copy each to its real location and edit the placeholder SSID/ passphrase values (each marked **CHANGE THIS** directly in the file, including the specific syntax gotchas for that file — see ResearchJournal.org for a real case that bit us):

Example file	Real location
<code>hostapd.conf.example</code>	<code>/etc/hostapd/hostapd.conf</code>
<code>dnsmasq.conf.example</code>	<code>/etc/dnsmasq.conf</code>
<code>wpa_supplicant.conf.example</code>	<code>/etc/wpa_supplicant/wpa_supplicant.conf</code>

`hostapd.conf.example` uses `driver=nl80211` and CCMP-only encryption (the modern, correct choices for this chip and for current security practice — not the older `driver=hostapd`~TKIP combination some dated guides still show). The AP's own address, `~192.168.4.1/24`, is chosen specifically to avoid colliding with the `172.16.42.0/24` range already used for the USB network link.

The two files' passphrase fields use different, easy-to-confuse conventions if editing both in one sitting: `hostapd.conf` does **not** use quotes and treats `#` as a comment marker (so a `#` anywhere in the passphrase silently truncates it, and adding quotes out of habit makes them part of the literal passphrase); `wpa_supplicant.conf` **does** need quotes around `ssid` and `psk`. Both require 8-63 ASCII characters (WPA2-PSK's own hard requirement).

2.8.3 Switching

`dudhya-wifi-mode` (install to e.g. `/usr/local/bin/`) handles the actual switch — unconditionally stopping every service either mode could have running, resetting the interface, and starting only the target mode’s services:

```
sudo dudhya-wifi-mode station
sudo dudhya-wifi-mode ap
```

Deliberately unconditional (stops everything every time, even if the target mode is already active) rather than only stopping the **other** mode’s services: re-running this after only editing a config file (e.g. a new passphrase) must force a real restart, not a no-op — OpenRC’s own **start** on an already-started service does not reload it, which would otherwise silently leave a stale, previously-loaded value in memory despite the file having changed. See ResearchJournal.org for the real case this fixes.

Not wired into OpenRC’s own boot sequence yet — run manually for now.

2.8.4 Open items

- **Station mode confirmed working:** switched cleanly, associated, obtained a DHCP lease, connectivity confirmed.
- **AP mode authentication: confirmed working** — an earlier restart-sequence bug in `dudhya-wifi-mode` (not reloading already-running services) was the actual cause of a client- authentication failure; fixed and re-tested directly.
- **Power-save mitigation confirmed helpful for raw connectivity** — an earlier idle-link instability (periodic packet loss, eventual re-authentication failure) no longer occurs with `dudhya-host-connect radio`’s `power-save-disable` mitigation in place.
- **Accepted limitation: sustained heavy traffic (e.g. package downloads) stalls while AP mode is active** — confirmed via `dudhya-device-update` in **both cable** and **radio** mode: traffic stalls and times out after a few minutes whenever the device’s AP is running, regardless of which interface is actually being provisioned over. That last point is the key clue: a WiFi-link-quality problem could not explain a **cable-mode** stall, but a driver/kernel-level stall in the WiFi subsystem could affect the whole system’s networking while active. Matched against real, documented reports of exactly this category of issue in `brcmfmac` specifically (the same driver family as this device’s BCM4339) — kernel-level command-queue stalls under active AP-mode load, a different mechanism from the earlier idle-link power-save issue, found in more than one independent report. **Not pursued as a fix** — genuine driver/kernel-level territory, outside this project’s reasonable scope, matching the precedent already set for the poweroff limitation. **Practical guidance:** prefer **cable** mode for provisioning/heavy transfers; if using AP mode, avoid sustained heavy traffic while it’s active, or switch to **station** mode (or stop the AP) first. AP mode’s

own intended purpose — a lightweight MQTT hub for other in-vehicle devices — involves much lighter, less sustained traffic than package downloads, so this limitation may not matter much in practice for the actual deployment.

- Concurrent station+AP operation was not pursued (see above); revisit only if a real need for it comes up, and check the chip’s actual supported interface **combinations** (not just supported modes) before attempting it.
- No internet-sharing/NAT setup for AP-mode clients — this serves the project’s actual need (other in-vehicle devices reaching the phone’s own MQTT broker), not necessarily giving them internet access through the phone. Add **iptables MASQUERADE** rules later if that turns out to be needed too.

2.9 Provisioning: scripted package installation/upgrade

Automates the manual steps in "Post-boot: internet access and package management" above and their WiFi-radio equivalent, as a matched pair of scripts — one for the host, one for the device — covering both connection types *dudhya* devices support. **cable mode confirmed working against the physical unit** — see below. **radio** mode’s own interface-detection logic is also confirmed correct, but an actual package operation over **radio** stalls whenever the device’s own AP is active — see the WiFi section’s own "Open items" for this same, accepted driver- level limitation, not a Provisioning-script bug.

2.9.1 On the host: **dudhya-host-connect**

Run on the development machine (laptop). Auto-detects what it needs rather than requiring fixed interface names, since neither connection type has one that’s stable (the USB gadget’s name is MAC-derived and changes per phone reboot; which WiFi interface joined the device’s AP isn’t known in advance either):

```
sudo dudhya-host-connect cable
sudo dudhya-host-connect radio
```

cable detects the USB gadget interface from **dmesg** (the same "renamed from usb0" line this project has relied on for diagnosis throughout), assigns it **172.16.42.2/24**, and NATs it out through whichever interface currently holds the host’s default route (also auto-detected, not assumed). **radio** expects this host to have **already joined the device’s own WiFi AP** manually (via the host’s normal WiFi settings — scripting that part was judged too host-setup-specific to automate reliably); detects which local interface holds an address in **192.168.4.0/24** and NATs that the same way. **radio** also disables power-save on that interface (**iw dev <iface> set power_save off**) — a mitigation for a real, matched connectivity- stability issue; see the WiFi section’s "Open items" and ResearchJournal.org for the full reasoning. Not fatal if it fails (older ~iw~/drivers may not support the setting).

Confirmed working, **cable** mode: correctly detected a fresh, previously-unseen USB gadget interface name (**enxeeae0ad470c**) purely from **dmesg**, and correctly detected the

host's own internet interface (`enp0s31f6`) — both exactly as designed, on the actual hardware, not just against fake test input.

2.9.2 On the device: `dudhya-device-update`

Run on the phone itself, after the host-side script above (or after confirming the equivalent is already active):

```
sudo dudhya-device-update cable
sudo dudhya-device-update radio
sudo dudhya-device-update cable mosquitto-dev build-base
```

Sets a default route and DNS for whichever mode, then runs `apk update` plus either `apk upgrade` (no extra arguments) or `apk add` (arguments passed straight through as package names). `radio` reads the actual current gateway from `dnsmasq`'s own lease file (`/var/lib/misc/dnsmasq.lease`) when available, since that is the accurate, current answer rather than an assumption; falls back to `192.168.4.2` (the first address in `dnsmasq.conf.example`'s DHCP range) if that file is missing or empty, which is a reasonable default only if this host is the only one that has ever connected.

Confirmed working, `cable` mode: both a plain `apk upgrade` (six packages upgraded) and `apk add` with an explicit package list (used to install the `hostapd`~/~`dnsmasq`~/~`wpa_supplicant` packages needed for the WiFi work) completed successfully against the real device.

2.9.3 Is cable and radio mutually exclusive?

Asked directly — the honest answer has two parts, not one:

- **At the hardware/interface level, no.** `usb0` and `wlan0` are entirely separate physical interfaces; nothing stops both being up at once. Different from the station-vs-AP question, which **was** a genuine single-radio constraint (one WiFi chip, one mode at a time) — USB and WiFi don't share hardware the same way.
- **For which one carries the device's own outbound internet traffic, effectively yes — but only because of how `dudhya-device-update` is written, not anything fundamental.** It uses `ip route replace default`, so running it again in the other mode replaces the default route rather than adding a second one alongside it. Direct reachability over either link (SSH, MQTT) should keep working regardless, since neither needs the default route on its own directly-connected subnet.

2.9.4 Known quirk: USB not detected if the cable is connected after boot, not before

Confirmed directly: booting the device with the cable already connected works reliably (matching every session so far in this project); booting without it and connecting afterward does not — nothing relevant appears in the host's `lsusb` or `dmesg`. This matches a real,

documented category of issue on mainline-ported phones generally: USB gadget binding (via configs, to the actual hardware UDC) has known timing sensitivities around whether a host is already present at the exact moment that binding happens during boot — confirmed via a real historical kernel-mailing-list patch for a **different** device (N900, interestingly the opposite direction: that one failed when the cable **was** attached during boot) and a separate real postmarketOS issue for yet another device showing the same general category of boot-time USB-gadget fragility. No hammerhead-specific report found matching this exact direction; this is a well-grounded hypothesis, not a confirmed root cause. **Practical workaround, already in use and good enough:** always connect the cable before powering on. Same spirit as the poweroff situation — a low-cost habit, not worth kernel-level investigation for something this narrow.

2.9.5 Known quirk: host reboot can assign the wired ethernet and USB gadget the same interface name

Observed directly: after a host reboot, the wired (RJ45) interface and the USB gadget interface ended up with the exact same name, and the two were mutually exclusive (only one usable at a time). Not independently confirmed, but the likely mechanism is udev's own predictable-naming fallback behaviour: when a stable, path-based name (like `enp0s31f6`) can't be determined for some boot-timing reason, udev falls back to a MAC-derived name (`enx<mac>`) — if this happened to the wired interface on this particular boot, a collision with the USB gadget's own MAC-derived name becomes possible. `dudhya-host-connect` already refused to proceed in the equivalent situation for `radio` mode; the same check has now been added to `cable` mode too, since this observation confirms it's a real risk there as well, not just a hypothetical one. If this happens: rebooting the host again, or checking `ip a` manually, are the practical things to try.

2.9.6 Open items

- `radio` mode's own interface-detection logic is confirmed correct against real hardware, but an actual package operation over `radio` stalls whenever the device's own AP is active — the same accepted driver-level limitation as the WiFi section's own AP-mode sustained-load finding, not a Provisioning-script bug. `cable` mode is unaffected and is the confirmed, working path for anything beyond light traffic.
- A real bug was caught and fixed during development, worth knowing about if these scripts are ever modified: under `set -e`, a failed detection function used directly in a command-substitution assignment (e.g. `iface="$(detect_something)"`) aborts the whole script immediately at that line, **before** the script's own intended "not found, here's what to check" error message can run. Confirmed directly (a minimal reproduction, not just reasoned about) before fixing every such call site with `"$(... || true)"`. See ResearchJournal.org.

- **radio** mode's exact topology (host joins the device's own AP and relays its own separate internet connection back, mirroring the cable-mode role split) is this project's own interpretation of "the necessary...for package installation," not independently confirmed — flagged directly rather than assumed silently correct.

2.10 Precedent

Two existing projects on this exact device are worth following rather than re-deriving from scratch:

- [in github masneyb/nexus-5-upstream project](#) — the upstream mainline-kernel port this procedure relies on; demonstrates display, GPU, and sensors working under a full desktop session.
- [in github Bry50/Old-Smartphone-as-a-server project](#) — a Nexus 5 running postmarketOS repurposed as a Docker/Home Assistant host, i.e. close to dudhya's own "embedded system, source of telemetry" goal already, and the source of the install-command sequence above.

3 **TODO** Google Pixel 3a (google-sargo)

Second device, chosen pragmatically: already owned, sidestepping the need to source one. Already covered as a ranked alternate (not the top pick, but a real, researched candidate) in "Annex: Second-device selection research" and its extended pass — this section pulls the relevant findings forward into an actionable procedure, rather than re-deriving them; see those annexes for the full sourced detail behind every claim below.

3.1 Target

Google Pixel 3a, codename **google-sargo**, Snapdragon 670 (SDM670). postmarketOS "community" tier. Smallest and lightest of every candidate researched (151.3 x 70.1 x 8.2mm, 147g) — an advantage if mounting space in the eventual vehicle deployment is tight.

3.2 Prerequisites

Not yet verified on the actual unit — do not assume this matches the Nexus 5's own experience, which had no "OEM unlocking" toggle at all and needed a direct **fastboot oem unlock**. Google's own Pixel line is generally documented as simpler: enable Developer Options, toggle "OEM unlocking" there, then **fastboot flashing unlock** (note: **flashing unlock**, not **oem unlock** — a different subcommand on Pixel devices per Google's own documentation). **Confirm this directly on the actual device before relying on it** — the Nexus 5's own procedure taught this project not to trust unlock documentation until physically checked.

3.3 Procedure (planned, not yet attempted)

```
pmbootstrap init      # select google-sargo when prompted
pmbootstrap install
pmbootstrap flasher flash_kernel
pmbootstrap flasher flash_rootfs
```

Matches the general postmarketOS install pattern already confirmed working for the Nexus 5, substituting the device codename — the exact flashing sequence for Pixel devices (fastboot-based, similar in spirit to hammerhead’s own lk2nd/ fastboot flow) needs confirming hands-on rather than assumed identical.

3.4 Known caveats (from prior research — expect these, don’t discover them blind)

- **GPS needs manual intervention on every boot, no A-GPS:** “the engine lock prevents the device from getting a useful position and the modem only outputs proprietary GPS NMEA messages,” requiring

```
qmicli -d qrtr://0 --loc-set-engine-lock=mt
qmicli -d qrtr://0 --loc-set-nmea-types=all
```

re-applied every boot (see the second-device annex for the full source). Without A-GPS, cold-start first fix is slow — plan for this, don’t assume quick GPS lock during testing.

- **ADSP can hang on resume from suspend:** FastRPC continuing across suspend crashes the ADSP; a pre-suspend hook stopping FastRPC via elogind is the documented fix (pmaports #3052). If this device is ever suspended (not just powered down) as part of the eventual deployment, confirm this hook is in place.
- **WiFi 5GHz reported not working in Europe** (pmaports #3134) — directly relevant given this project’s own location; test both bands rather than assume 5GHz works.
- CPU thermal throttling and an unsupported Titan M/secure element are also noted in prior research — lower priority for this project’s headless telemetry use case, but worth knowing about.

3.5 Sensor inventory (not yet tested)

Expected via the merged sensor-support framework (MR !5329, ADSP- based rather than SLPI — SDM670 shares the ADSP for sensors rather than having a dedicated SLPI, per the second-device annex). To confirm once running, matching the same physical-test rigor already applied to the Nexus 5 (real values, not just “device enumerated”):

Sensor	Expected	Confirmed
Accelerometer	yes	not yet
Gyroscope	yes	not yet
Magnetometer	yes	not yet
Light	yes	not yet
Proximity	yes	not yet
Barometer	unknown	not yet
GPS/GNSS	yes, with the manual steps above	not yet

3.6 Power management: running without a battery

See FutureResearch.org's "Power a candidate device without its battery" item for the general framework this fits into.

Confirmed, device-specific procedure found: three sources (two YouTube videos plus an Instructables writeup) are all the same author, SquareOrbits, and — a genuine stroke of luck — their own demo device **is** a Pixel 3a, not a different phone needing adaptation ([Instructables, "Power a Phone Without a Battery"](#) — see ResearchJournal.org).

Approach, and why it's better-suited to this device than the generic technique below: rather than build a "fake battery" from discrete parts, this method keeps the phone's **own** original battery protection board and only replaces the cell:

1. Disassemble the phone (the source used [iFixit](#)'s own Pixel 3a guide) and remove the battery.
2. The battery has two parts — a bare cell, and "a long, thin circuit board that protects and monitors it." Desolder the board from the cell and discard the cell. The board alone "expects to see a voltage roughly between 3V to 4.2V depending on the state of charge" — feed it that range directly and the phone boots normally. This sidesteps needing to separately emulate a thermistor reading from scratch (the general technique below does need that) — the retained board already handles whatever sensing the phone's own PMIC expects.
3. Power source: a USB-C Power Delivery trigger module set to request 9V, feeding an XL4015 buck converter adjusted to output 4-4.2V, connected to the retained board's (+)/(-) tags (polarity matters).
4. **Critical, empirically confirmed step:** run a separate wire from the board's negative (-) tag to the phone's own chassis ground (e.g. under the USB-C port's mounting screw). The source states plainly: "Without this the phone may not boot."
5. Reassemble, connect a PD-capable power bank/charger — boots normally.

BOM coherence, checked as requested: all four listed parts are real, well-matched, and appropriately rated for this specific purpose — the XL4015 is a genuine, common buck-converter IC (input up to ~36V, several amps output, comfortably handles 9V-to- ~4.2V at

phone-level currents); a USB-C PD trigger module is the correct way to get a stable voltage above 5V for the buck converter to regulate down from with good margin; a PD-capable power source is the necessary counterpart. No mismatched or inappropriate parts found.

Discrete-component KiCad schematic: see `dudhya-pixel3a-battery-free-power.kicad_sch` (plus its matching `dudhya-pixel3a-battery-free-power.kicad_pro` project file), regenerated via `make kicad-regen` (see the project's own README for build prerequisites, including packpay). Builds the equivalent of the trigger-module + buck-converter pair from real, individually sourced parts rather than pre-made boards: an STUSB4500 (ST's own dedicated, autonomous USB-PD sink controller IC, configured to request a fixed 9V with no microcontroller needed at runtime) driving a P-MOS power-path switch, feeding an XL4015-based buck converter matching the source's own design, plus a reverse-current protection diode (D2) on the output. Every symbol is copied verbatim from KiCad's own bundled libraries, and the STUSB4500 portion follows ST's own official "minimum implementation, fixed PDO, no MCU" reference circuit from its own datasheet. Primary sources: [STMicroelectronics, STUSB4500 product page](#); [STUSB4500 datasheet, DS12499 Rev 8](#) (the "minimum implementation, fixed PDO, no MCU" reference circuit and the pin table's own "typical connection" column, both used directly below); [STL6P3LLH6 product page](#) and [STL6P3LLH6 datasheet, DS9257 Rev 4](#) (the P-MOS power-path switch, also part of ST's own reference design — 30V/6A/24mOhm typ., comfortably rated for this application).

The schematic is electrically verified: zero ERC errors, all real components carry a genuine, researched footprint (not a placeholder), and every component's own position is decided algorithmically from the real netlist rather than hand-placed (see DevelopmentStandard.org for how this generation pipeline works). The previous, hand-placed version of this same circuit is kept for reference at `dudhya-pixel3a-battery-free-power-handplaced.kicad_sch` (`make kicad-handplaced-regen`) and remains the real netlist source the promoted generator itself reads from.

Reverse-current protection on the output, per the source's own follow-up video: a diode on the buck converter's output ensures the phone cannot back-feed current into the buck converter, relevant if something else is later plugged into the phone and its own charging circuitry tries to draw or push current the other way. **Practical consequence worth flagging:** a series diode drops some voltage (a Schottky, as used here, drops less than a standard silicon diode — roughly 0.3-0.4V rather than 0.6-0.7V), so the output trimmer must be tuned by measuring voltage **at the phone's own connector, after D2** — not at the buck converter's raw output pin — or the phone ends up slightly under-fed.

Breadboard-style companion diagram: a second, connectivity- focused rendering of this same circuit, styled to read like a breadboard rather than a formal schematic. Explicitly a visualization aid, not a literal build guide (the STUSB4500 and XL4015 above are not themselves breadboard-native packages). Generated by `build_pixel3a_breadboard_autolayout.py` (`make breadboard-verify`) — every component's own position and every wire derived directly from the real, loaded netlist, not hand-placed. A previous, hand-placed version remains available, unchanged, as `build_pixel3a_breadboard.py` (`make breadboard-handplaced-verify`) for reference and comparison.

3.6.1 Complete wiring specification (matches the now-verified schematic file)

Parts list:

Ref	Part	Notes
U1	STUSB4500QTR	USB-PD sink controller
U2	XL4015	Buck converter
J1	USB-C receptacle, power-only	VBUS/GND/CC1/CC2 only needed
Q1	P-MOSFET (STL6P3LLH6 or eq.)	Power-path switch
D1	SS34 (3A Schottky)	Buck converter's own freewheel diode
D2	Schottky, e.g. SS14 (1A+)	Output reverse-current blocker
L1	33uH inductor	Rated for expected load current
R1	10k	RESET pulldown
R2	10k	Q1 gate pull-up
R3	1k	FB divider, bottom leg
RV1	2.2k trimmer potentiometer	FB divider, adjustable
C1	1uF	VREG_1V2 decoupling
C2	1uF	VREG_2V7 decoupling
C3	100uF	Buck input cap
C4	220uF	Buck output cap
C5	2.2nF	VC compensation (typical, may need tuning)

Not a physical part: the schematic also carries three **PWR_FLAG** markers (a standard KiCad software convention, satisfying ERC's requirement that externally-fed power nets be explicitly marked as such) — nothing to buy or solder.

Wiring, by net — every connection needed, derived directly from ST's own datasheet ("typical connection" column), and now confirmed to match the actual schematic file via clean ERC:

- **GND:** J1 (A12, B12, SHIELD) — U1 (pin 10, pin 25, VSYS/pin 22, ADDR0/pin 12, ADDR1/pin 13, CC1DB/pin 1, CC2DB/pin 5) — R1 (pin 2) — C1 (pin 2) — C2 (pin 2) — C3 (pin 1) — C4 (pin 1) — C5 (pin 2) — D1 (Anode) — R3 (pin 2) — U2 (GND/pin 1) — the phone's own battery PCB (-) tag — **and** a separate wire to the phone's chassis (both required, per the confirmed procedure above).
- **VBUS_9V** (raw negotiated 9V, before the switch): J1 (VBUS: A9, B9) — U1 (VDD/pin 24, VBUS_VS_DISCH/pin 18) — Q1 (Source) — R2 (pin 1).
- **CC1:** J1 (A5) — U1 (CC1/pin 2) — direct, no other nodes.
- **CC2:** J1 (B5) — U1 (CC2/pin 4) — direct, no other nodes.
- **RESET pulldown:** U1 (RESET/pin 6) — R1 (pin 1).
- **PMOS gate:** Q1 (Gate) — R2 (pin 2) — U1 (VBUS_EN_SNK/pin 16).

- **Switched output, into the buck:** Q1 (Drain) — C3 (pin 2) — U2 (VIN/pin 5).
- **Buck’s own switching node:** U2 (SW/pin 3) — L1 (pin 1) — D1 (Cathode).
- **VC compensation:** U2 (VC/pin 4) — C5 (pin 1).
- **Regulated output, before D2:** L1 (pin 2) — C4 (pin 2) — RV1 (pin 1) — D2 (Anode).
- **FB divider midpoint:** RV1 (pin 2) — R3 (pin 1).
- **Feedback:** RV1 (wiper/pin 3) — U2 (FB/pin 2).
- **To the phone** (after D2 — this is what actually reaches it): D2 (Cathode) — the phone’s own battery PCB (+) tag.

Left unconnected in this minimal build (no MCU, no status indication): U1’s SCL (pin 7) and SDA (pin 8) are proper no-connects in the file — to reprogram the requested PDO later, temporarily remove those and attach an external I2C programmer; ATTACH (pin 11), ALERT (pin 19), GPIO (pin 15), POWER_OK2 (pin 20), POWER_OK3 (pin 14), A_B_SIDE (pin 17), DISCH (pin 9), NC (pin 3) are also no-connects.

General technique, kept as useful background (for a device where the original board can’t be reused, e.g. if it’s damaged): a from-scratch “fake battery” circuit, corroborated by an independently-found technical write-up (Yaky, “Smartphone without a battery”, yaky.dev, 2022-09-06 — a different device, a Galaxy S5) — a diode for voltage drop and reverse protection, a resistor emulating the thermistor reading, a bulk capacitor for load transients. Not needed for the Pixel 3a specifically now that the device-specific, board-preserving method above is confirmed, but worth keeping on file for a future device without a reusable board.

4 Annex: Second-device selection research

Context: the Nexus 5’s GPS is confirmed dead (modem subsystem non-functional — see the “Power management” and sensor-inventory sections above), and the originally-planned second device, the Nexus 7 2013, most likely never had GPS hardware at all on its WiFi-only variant (`asus-flo`) — GPS on that product line was tied to the LTE-cellular variant specifically, per its own published specifications, following the same “GPS rides on the modem” pattern already confirmed on the Nexus 5. Rather than repeat that mistake — picking a device on general reputation and discovering its GPS status only after a multi-session resuscitation effort — this annex is a thorough, sourced comparison across postmarketOS’s actual device pool, produced before committing to a second device. Full citations preserved throughout, consistent with this project’s own practice of never asserting a technical claim without a source.

4.1 Summary and recommendation

Top pick: Xiaomi Poco F1 (xiaomi-beryllium, 2018). Best- balanced choice: GPS/GNSS works under postmarketOS (via ModemManager + Geoclue, documented to work even without a SIM), the full sensor suite works through the merged SDM845 Snapdragon Sensor Core framework, it sits in postmarketOS’s ”community” tier (the highest tier with any real population — ”main” is currently essentially empty outside QEMU targets), and it is the cheapest serious candidate (~\$90-160 used).

Alternates: OnePlus 6 (oneplus-enchilada, 2018) — near- identical SDM845 hardware and sensor support, but GPS reliably works only if OxygenOS 9.0.8 is flashed first; and **Google Pixel 3a** (google-sargo, 2019) — sensors and GPS both work, but GPS needs a `qmicli` engine-lock reset re-applied on every boot and has no A-GPS, making first fix slow and headless operation more fragile.

Not recommended for this project: Fairphone 4 (GPS only partially confirmed under postmarketOS, mobile data broken, newer/pricier), Samsung Galaxy S9 ”starlte” (the community-tier port is the Exynos variant, with broken WiFi/Bluetooth), and Nokia N900 (GPS confirmed working, but 2009 hardware — 600MHz single- core, 256MB RAM, no gyroscope/magnetometer/barometer — is too weak for this project’s actual sensor-telemetry workload).

Extended criteria (dimensions/weight, battery-free operation feasibility, cybersecurity/blob posture; see the three new sections below and ResearchJournal.org for the full reasoning): the Poco F1 **still holds up** as the top pick overall. But one real, newly-surfaced nuance changes the practical plan: **the OnePlus 6/6T are the only candidates with a documented, reproducible method for booting without a battery at all** (their pack carries its own smart-battery fuel-gauge chip, which the Poco F1’s pack does not). Given the project’s own separate, already-planned interest in battery-free operation (see ”Powering the device without its battery” in FutureResearch.org), the practical plan is now: **Poco F1 for the production node, OnePlus 6 or 6T as a dedicated battery-free R&D unit** to prototype and de-risk that work before attempting it on the Poco F1 itself, where it remains unproven rather than confirmed infeasible.

4.2 Ranking table

Rank	Device	Codename	Tier	GPS under pmOS	Price
1	Xiaomi Poco F1	xiaomi-beryllium	community	Works	~\$90-160
2	OnePlus 6	oneplus-enchilada	community	Works (a)	~\$120-170
3	Google Pixel 3a	google-sargo	community	Works (b)	~\$90-150
4	OnePlus 6T	oneplus-fajita	community	Works (a)	~\$120-180
5	Fairphone 4	fairphone-fp4	community	Unconfirmed	~\$200+
	Samsung Galaxy S9	samsung-starlte	community	Unconfirmed	~\$90-140
	Nokia N900	nokia-n900	community	Works (c)	~\$60-150

(a) needs a specific OxygenOS version pre-flashed first, see below. (b) needs a `qmicli` reset re-applied every boot, no A-GPS, see below. (c) works, but the 2009 hardware lacks a gyroscope, magnetometer, and barometer — see below. Sensor completeness for every device is covered in the per-device findings that follow, not repeated here to keep this table scannable.

4.3 Per-device findings

4.3.1 Xiaomi Poco F1 (`xiaomi-beryllium`) — top pick

SoC: Qualcomm Snapdragon 845, the best-supported mainline Qualcomm platform in postmarketOS. GPS works: the device wiki documents activating GPS via ModemManager’s CLI plus Geoclue, including a no-SIM path — valuable for a fixed telemetry node with no cellular plan ([postmarketOS wiki](#), [Poco F1](#)). Independently confirmed working by Drew DeVault’s own 2022 review: GPS itself works fine, only some map applications render it poorly ([A review of postmarketOS on the Xiaomi Poco F1](#)). Caveat: only XTRA A-GPS assistance is supported, not SUPL, so cold-start first fix can be slow without assistance data.

Sensors: full suite (accelerometer, light, proximity, compass) via the Snapdragon Sensor Core/SLPI framework, merged in pmaports (!4050, “[Qualcomm Snapdragon Sensor Core support for SDM845 devices](#)”); gyroscope runs on the same stack but isn’t individually confirmed in the merge notes — worth verifying on the actual unit, matching this project’s own established practice, rather than assumed from the framework alone.

Known issues: mobile data historically needed a specific stock firmware pinned before flashing; camera now works with a recent kernel; SDM845 suspend/power management is imperfect (a pattern this project has already seen with the Nexus 5’s own poweroff limitation, worth checking early rather than assumed away).

Price: cheapest serious candidate, ~\$90-160 used and widely available; one community user reported paying 150 CAD ([Sxmo on the poco f1](#) — [Anjan’s Homepage](#)).

4.3.2 OnePlus 6 (`oneplus-enchilada`) — strong alternate

Same SDM845 platform and sensor framework as the Poco F1. GPS works, but with a real prerequisite: the wiki states OxygenOS 9.0.8 (OP6) or 9.0.16 (OP6T) should be flashed to both slots before installing postmarketOS to get GPS working ([postmarketOS wiki](#), [OnePlus 6](#)). Sensors confirmed working via the same !4050 merge. Camera only partially works; fingerprint reader doesn’t work; the modem stack is described as “not particularly stable.” Plentiful secondhand, ~\$120-170.

Functionally equivalent to the Poco F1 for telemetry purposes, but the mandatory OxygenOS pre-flash for guaranteed GPS is an extra provisioning step and a source of fragility the Poco F1 doesn’t need. Worth choosing over the Poco F1 only if the latter isn’t available at a reasonable price.

4.3.3 Google Pixel 3a (**google-sargo**) — alternate with a GPS caveat

SoC: Snapdragon 670. GPS is supported but needs manual, per-boot intervention: the wiki documents that a hardware "engine lock" prevents a useful position by default, requiring `qmicli -d qrtr://0 --loc-set-engine-lock=mt` and `--loc-set-nmea-types=all` to be re-applied **on every boot**, and notes there is no A-GPS so first fix can take a long time ([postmarketOS wiki](#), [Google Pixel 3a](#)). Scriptable (this project already has the habit of wrapping exactly this kind of per-boot setup into a startup script, e.g. `dudhya-sensord`'s own gyroscope auto-zero), but a real, structural piece of extra fragility compared to the SDM845 devices.

Sensors: added via a separate merged MR ([pmaports !4050](#) covers SDM845; Pixel 3a's own sensor support was merged separately for its SDM670/ADSP stack), covering accelerometer, gyroscope, compass, and proximity.

Known issue worth flagging directly: the wiki documents that the ADSP can crash and hang the device on resume from suspend unless a pre-suspend hook stops FastRPC first — already handled by a default `elogind` hook, but worth checking if this project's own minimal console setup uses `elogind` at all before assuming this is covered ([postmarketOS wiki](#), [Google Pixel 3a](#)). 5GHz WiFi reported not working in some regions. Cheap and common, ~\$90-150 used.

4.3.4 OnePlus 6T (**oneplus-fajita**)

Same SDM845 platform, tier, and sensor support as the OnePlus 6, GPS works with the OxygenOS 9.0.16 pre-flash — but audio is broken (different speaker hardware lacking driver support, no headphone jack). Audio is irrelevant to a telemetry node, but the 6T offers no advantage over the plain OnePlus 6 for this project's purposes while costing about the same or more — no reason to prefer it.

4.3.5 Fairphone 4 (**fairphone-fp4**) — not recommended

SoC: Snapdragon 750G. Sensors now supported via a merged MR ([!5290](#), "Add sensor support for Fairphone 4"). GPS status under postmarketOS specifically could not be confirmed with a direct source — treated as unconfirmed, not assumed working, matching this project's own standard for GPS claims after the Nexus 5 experience. More seriously: mobile data is broken ([pmaports #2550](#), "Modem / mobile data on Fairphone 4"), and speaker/microphone don't work. Released 2021, discontinued 2024; secondhand price meaningfully higher (~\$200+) than the 2018 candidates. Unconfirmed GPS plus broken mobile data plus a higher price rules it out as the second device for now.

4.3.6 Samsung Galaxy S9 (**samsung-starlte**) — not recommended

The community-tier port is specifically the Exynos 9810 variant, not Snapdragon — on this port, WiFi and Bluetooth do not work at all (a Broadcom chip needing driver work not yet done), with reported display bring-up issues too. A separate Snapdragon variant

effort exists but reportedly fails to start the display around half the time and also lacks WiFi/Bluetooth. GPS status under postmarketOS is unconfirmed. Broken WiFi alone is disqualifying for a device meant to relay telemetry.

4.3.7 Nokia N900 (**nokia-n900**) — GPS confirmed, hardware too weak

GPS is genuinely confirmed working, via the `gps-nokia-n900` package (sourced from the `maemo-leste` project), exposing an NMEA stream directly. A real, useful data point. But the hardware (2009, 600MHz single-core TI OMAP3430, 256MB RAM) has only an accelerometer, proximity, and light sensor — no gyroscope, no magnetometer, no barometer — failing this project’s own sensor- completeness bar (the Nexus 5 alone already provides all of these). Interesting as a “GPS works” reference point, not suitable as the actual second device.

4.4 Physical dimensions and weight

For in-vehicle mounting planning. Source: manufacturer specs and GSMArena.

Device	Codename	H x W x D (mm)	Weight
Xiaomi Poco F1	xiaomi-beryllium	155.5 x 75.3 x 8.8	182 g
OnePlus 6	oneplus-enchilada	155.7 x 75.4 x 7.75	177 g
OnePlus 6T	oneplus-fajita	157.5 x 74.8 x 8.2	185 g
Google Pixel 3a	google-sargo	151.3 x 70.1 x 8.2	147 g
Fairphone 4	fairphone-fp4	162 x 75.5 x 10.5	225 g
Samsung Galaxy S9	samsung-starlte	147.7 x 68.7 x 8.5	163 g
Nokia N900	nokia-n900	110.9 x 59.8 x 18	181 g

The top three candidates cluster tightly (~156-158mm tall, 177-185g) — size/weight doesn’t meaningfully distinguish between them. The Pixel 3a is notably smaller and lighter (147g), an advantage if mounting space is tight. The Fairphone 4 is the heaviest and thickest (225g, 10.5mm) — the cost of its user-serviceable design. The N900 is short but unusually thick (18mm+) because of its slide-out keyboard, an awkward shape for flat mounting — moot, since it’s already ruled out on other grounds.

4.5 Battery-free (external DC) operation feasibility

Device-specific characteristics only — the general method, procedure, tools, and bill of materials for battery elimination itself is a separate, not-yet-started research item (see [FutureResearch.org](#), “Power the device without its battery”).

The deciding factor is **where the fuel-gauge chip lives**:

- **OnePlus 6 and 6T**: the battery pack itself carries a [TI BQ27411](#) fuel gauge, communicating with the phone over I2C on the battery flex — confirmed by the actual postmarketOS upstreaming patch enabling it ([Caleb Connolly](#), “[arm64: dts: qcom:](#)

[sdm845-oneplus-*: add fuel gauge](#)). This on-pack "smart battery" design is **why** a documented batteryless-boot method exists for the 6T: reuse the pack's own BMS board plus a MUR460 diode to spoof battery presence, a technique credited to the GreatScott! YouTube channel and documented working under postmarketOS specifically ([Kédio Blog, "Running a OnePlus 6T without battery \[Smartphone Revival Project\]"](#)). This is the single best-documented batteryless-boot case found across every candidate.

- **Poco F1 and Pixel 3a:** the fuel gauge is integrated into the mainboard PMIC instead ([kernel patch series, "Qualcomm 3rd gen fuel gauge support"](#), confirming this specifically for both `sdm845-xiaomi-beryllium` and `sdm670-google-sargo`), and their packs are conventional cell + protection + NTC thermistor with no data line. Likely **feasible** (a real battery voltage plus a correctly-valued NTC resistor should satisfy the PMIC) but **unproven** — no published bench-supply boot was found for either. A real Poco F1 user report of a unit with a faulty battery "bootloop[ing] unless charger[/USB] plugged in" ([XDA Forums, "\[FIXED\] POCO F1 stuck in bootloop unless charger connected / plugged in"](#)) is evidence it can run on external power but is unstable without a valid battery/gauge reading present.
- **Fairphone 4 and Nokia N900:** both have user-removable packs (easiest mechanical access, no soldering/adhesive to fight), but neither has a documented postmarketOS batteryless-boot procedure. A removable pack still has protection/NTC (and, on the N900, a BSI/ID resistor) that a dummy source must satisfy.
- **Samsung Galaxy S9:** pack is glued in place (least accessible mechanically); no batteryless-boot documentation found.

Cross-cutting principle, from general charger-IC behaviour rather than a device-specific source: disconnecting the battery also disconnects the NTC thermistor, which reads as out-of-range and can block boot or charging unless the temperature-sense input is either spoofed with a resistor or the relevant safety check is bypassed — this is **why** a naive "just apply DC power" approach doesn't reliably work, and why the OnePlus 6T's published recipe specifically routes through the pack's own BMS rather than wiring straight to the cell contacts.

Practical implication for this project: acquire an OnePlus 6 or 6T specifically as a battery-free R&D unit, using the documented 6T method to prototype and de-risk the general technique, before attempting it on the Poco F1 (where it remains a reasoned guess, not a confirmed procedure).

4.6 Cybersecurity and binary-blob posture

postmarketOS installs non-free firmware by default when a device needs it to function — confirmed via the project's own blog post ("[Non-free firmware is installed by default](#)", 2024-02-15): "All pre-built postmarketOS images have always included non-free firmware."

Every Qualcomm candidate here needs some of it; the question is how much, and what it's exposed to.

Shared blob requirements, SDM845 trio (Poco F1, OnePlus 6/6T): Adreno GPU firmware, ath10k/WCN3990 WiFi firmware, modem/baseband firmware, and SLPI sensor-core DSP firmware (the same [Snapdragon Sensor Core](#) stack behind the sensor support covered above) — all extracted from a specific stock-ROM snapshot (MIUI for the Poco F1, OxygenOS for the OnePlus units). The Pixel 3a (SDM670) needs the same classes of blob, with its modem additionally hosting the sensor DSP. The Galaxy S9 "starlte" port's WiFi/ Bluetooth (a Broadcom chip) simply doesn't work at all rather than running on a blob, per the port maintainers — consistent with the earlier finding.

GPS confirmed to require an active modem blob, on every Qualcomm candidate — the same "GPS rides on the modem" pattern already established on this project's own Nexus 5. postmarketOS's own documentation confirms Qualcomm modems (explicitly naming the Poco F1 among others) "provide a QMI Location Interface to retrieve GNSS locations," managed through ModemManager ([postmarketOS wiki, GPS troubleshooting](#)) — meaning the modem/baseband firmware blob must be loaded and running for GPS to work at all, not an optional component.

Specific documented vulnerabilities, for context rather than alarm: "QualPwn" (CVE-2019-10538, CVE-2019-10539, CVE-2019-10540 — [Tencent Blade Team, disclosed at Black Hat USA / DEF CON 27, 2019](#)), an over-the-air WiFi-to-modem exploit chain affecting, per Qualcomm's own advisory, the Snapdragon 835/845 family (matching the Poco F1 and OnePlus 6/6T) among other chipsets. Separately, Adreno GPU firmware has had actively-exploited local privilege-escalation CVEs (e.g. CVE-2023-33106), added to CISA's Known Exploited Vulnerabilities catalog in December 2023 ([CISA, "CISA Adds Four Known Exploited Vulnerabilities to Catalog"](#)). Kr00k (CVE-2019-15126, a WPA2 decryption flaw) affects Broadcom/Cypress WiFi chips — the family used on the Galaxy S9 ([ESET, "Kr00k"](#)) — though moot there since WiFi doesn't work on that port anyway. No device-specific CVEs were found for the SLPI sensor-core firmware itself; absence of findings, not a claim of absence of risk.

Mitigating factor worth naming directly: these blobs run on physically separate co-processors (modem, GPU, WiFi chip, sensor DSP), not inside the main Linux kernel — postmarketOS's own sensor-core documentation (linked above) notes "the SoC hypervisor prohibits accessing the sensors directly," which is why QMI/ FastRPC exist as the interface at all. Real isolation, not a security guarantee.

Practical takeaway for this project's actual deployment (a local WiFi AP/MQTT hub, not a general internet-facing host): the WiFi/modem interface is the realistic remote attack surface: if a deployed node doesn't need WiFi client connectivity in the field, disabling it shrinks that surface without affecting GPS (which rides the modem/QMI path, not WiFi).

4.7 Purpose-built mainline phones (secondary by project ethos)

PinePhone, PinePhone Pro, and Librem 5 are all community tier with well-supported GNSS and sensors, but as already agreed when this research was scoped, they sit outside this project's stated ethos of resuscitating **abandoned Android hardware** specifically, and none of them is dramatically better on GPS/sensors than the Poco F1 — not enough to override that ethos. Noted for completeness, not pursued further here.

4.8 Recommended validation sequence, once a unit is acquired

Matching this project's own established discipline (verify directly, don't assume a spec sheet or a wiki page): before treating any candidate as confirmed, run through the same kind of check this project already did for the Nexus 5's own sensors — real values, real before/after physical tests, not just "the file exists."

1. Confirm sensors enumerate and respond to real stimuli (matching the Nexus 5's own IIO-based verification method already documented above), via `monitor-sensor~/~lsiio~/~iio-sensor-proxy` as appropriate for the target device's own sensor framework.
2. Bring up GPS following the device's own documented procedure (ModemManager CLI + Geoclue for the SDM845 devices; the per-boot `qmicli` reset for the Pixel 3a), and confirm an actual fix, not just that the service starts.
3. Script whatever startup steps GPS needs into a boot-time service, following the same pattern already established for `dudhya-sensord`'s own gyroscope auto-zero and `dudhya-wifi-mode`'s config-reload discipline.

4.9 Caveats

- postmarketOS's own wiki blocks automated fetching (an anti-bot challenge); several device pages were read via search snippets and a mirror rather than the live wiki tables directly. The strongest evidence throughout is the merged GitLab pmaports merge requests, which are authoritative regardless.
- Gyroscope is inferred working on the Qualcomm candidates (same underlying sensor framework as the confirmed accelerometer/ light/proximity/compass) but not individually confirmed in any merge note — verify directly on the actual unit before relying on it, per the validation sequence above.
- Fairphone 4 GPS status under postmarketOS specifically remains genuinely unconfirmed, not just under-documented — treat as "needs bench verification," not as a soft "probably works."
- Prices are approximate secondhand ranges as of the research date and will vary by region, storage tier, and condition.

- Across every Qualcomm candidate, only XTRA A-GPS assistance is supported (not SUPL), so cold first fix can be slow; likely acceptable for a vehicle node with a clear sky view, but worth measuring rather than assuming.
- Battery connector pin counts are only partially verified (the Poco F1's specifically comes from a low-quality aggregator, not a confirmed teardown photo) — verify physically before any battery-free work begins.
- The batteryless-boot method is documented only for the OnePlus 6T specifically; its applicability to the OnePlus 6, Poco F1, Pixel 3a, Fairphone 4, and N900 is a reasoned inference from shared hardware, not independently proven for each.
- Whether cutting the NTC thermistor line alone faults a given device (versus firmware simply ignoring temperature sense) was not confirmed per device — inferred from general charger-IC behaviour, not a device-specific source.
- The CVEs cited (QualPwn, Adreno, Kr00k) are platform/chip-level, disclosed years ago and largely patched in current firmware; their real-world exploitability against a stripped-down postmarketOS node (optionally with WiFi disabled) was not independently assessed here. "No CVE found" for the SLPI sensor DSP specifically is an absence of findings, not proof of safety.

5 Annex: Operating system choice re-evaluation

Context: postmarketOS was chosen at this project's very start over two other real candidates — LineageOS (rejected: still Android/Bionic/ART on a vendor-forked kernel, not GNU/Linux) and UBports/Halium (rejected: a genuine Linux userspace, but layered over the phone's own frozen vendor Android kernel through a compatibility shim, not a real mainline kernel) — specifically because it targets genuine mainline Linux kernel support, matching Concept.org's own stated goal of "an open source operating system, preferably GNU/Linux." Not a whimsical choice; see ResearchJournal.org. This annex is a proper comparison re-validating that choice, keeping the same framing. Full citations preserved throughout.

5.1 Summary and recommendation

postmarketOS remains the correct choice for dudhya — it wins on every one of the priority-ordered criteria below: it is the only genuine mainline-Linux-kernel-first project with (1) a rigorous, enforced per-device tier system, (2) the broadest coverage of old Android phones specifically, (3) a documented, reusable Qualcomm sensor/GPS stack directly relevant to this project's own IIO-sysfs daemon, and (4) a funded, multi-developer organization mitigating bus-factor risk. No alternative found beats it on these criteria.

Mobian (Debian-based, also genuinely mainline-kernel-first) is the closest real peer and worth keeping as a documented fallback — it shares the same SDM845 kernel work

as postmarketOS, but supports far fewer old Android phones and has thinner per-device documentation. **Mobile NixOS** and **Maemo Leste** are also genuinely mainline-first but too small/niche to be primary candidates.

Droidian and Ubuntu Touch (UBports) are confirmed Halium/vendor- kernel-based, not mainline — structurally the lower tier the original framing predicted, not excluded outright but not a fit for this project’s own stated goal. **Sailfish OS is confirmed proprietary** (closed UI, EULA-covered components, Halium-based vendor kernels underneath) and stays out of scope entirely, the same way LineageOS-family systems already are.

A concrete, separate finding worth acting on: this project’s own Nexus 5 GPS failure generalizes — GPS quality is device-specific, and the newer Snapdragon 845 generation (OnePlus 6/6T, Poco F1, already the recommended second device — see the annex above) has confirmed working GPS via ModemManager/QMI, materially better supported than the Nexus 5’s older MSM8974 platform. This reinforces, rather than changes, the existing second-device plan.

5.2 Candidate classification

5.2.1 Genuine mainline-Linux-kernel-first (primary tier)

- **postmarketOS** (CHOSEN, incumbent) — Alpine Linux based (musl, BusyBox, apk). Explicit project goal is upstreamed, mainline kernel support, not a repackaged Android base ([postmarketOS homepage](#)). As of February 2026 it ships its own generic mainline kernel packages (`linux-postmarketos-mainline`, `-stable`, `-lts`), described as “the postmarketOS equivalents to Alpine kernels such as `linux-stable` or `linux-lts`,” intended to work across many devices and collapse the historical per-device kernel-maintenance burden ([postmarketOS packaging docs](#), “Generic Kernels”).
- **Mobian** — Debian-based, mainline kernel by explicit policy: “we do not wish to use compatibility layers... We would prefer to use a compatible kernel” ([Mobian blog](#), “[Porting Mobian to New Devices](#)”). Supports the same SDM845 old-Android phones as postmarketOS (OnePlus 6/6T, Poco F1) and explicitly defers to the postmarketOS wiki as its own reference for whether a device can boot mainline — the closest real peer to postmarketOS found.
- **Mobile NixOS** — NixOS/Nix-based, mainline where possible, a tiered device list (supported/almost/won’t work/not supported) ([Mobile NixOS \(community fork\)](#)). Small device set.
- **Maemo Leste** — Devuan/Debian based, mainline-based kernels, FOSS-only by policy (reverse-engineers closed drivers rather than using them). Its own site describes it as still “in development phase” ([Maemo Leste homepage](#)), with a very small device set (Nokia N900 among them) and an open call for more developers.

5.2.2 Vendor-kernel/Halium-shim (structurally lower tier, not excluded outright)

- **Droidian** — confirmed Halium/libhybris-based by its own documentation: "Droidian is based on Debian and utilizes Halium to interact with Android devices"; it "relies on the Android Linux kernel and drivers," patching security fixes onto that frozen vendor kernel rather than running a real mainline one ([Droidian FAQ](#), [Droidian porting guide](#), ["Kernel compilation"](#)). Its own materials market this explicitly as a breadth-over-maintainability tradeoff against "mainline distributions like Mobian and postmarketOS" ([LINMOB](#), ["State of the Droidian Project"](#)). Alive (30+ "snapshot" releases as of
 1. but a small team.
- **Ubuntu Touch (UBports)** — also Halium-based: "Almost all modern Ubuntu Touch ports run on top of Halium... the Android Linux Kernel operates the device" ([UBports](#), ["Halium"](#)).

5.2.3 Excluded (Android-based or proprietary)

- LineageOS, GrapheneOS, CalyxOS, /e/OS — Android (Bionic/ART), not GNU/Linux, excluded for the same reason as the original decision.
- **Sailfish OS** — confirmed proprietary: closed Silica UI, EULA-covered components (roughly a fifth of the OS), running on Halium-based vendor kernels underneath ([Wikipedia](#), ["Sailfish OS"](#)). Fails both the open-source and the mainline-kernel requirement; excluded.
- Purpose-built mainline phones (PinePhone, PinePhone Pro, Librem 5) remain lower priority per this project's own established ethos (see the annex above) — not re-litigated here.

5.3 Criterion: genuine mainline-kernel model

The gating criterion. postmarketOS, Mobian, Mobile NixOS, and Maemo Leste all pass it. Droidian and Ubuntu Touch fail it (frozen vendor kernel via Halium). Sailfish fails both this and the open-source requirement. This is the same axis the original decision was built on, now re-confirmed against a wider candidate pool rather than just the two alternatives considered at the time.

5.4 Criterion: community strength

Measured the way postmarketOS's own tier system does it (real per-device maintainer requirements, not raw install-count marketing):

- **postmarketOS**: funded, multi-developer, with formal governance (Trusted/Core Contributor roles) and a written Contributor Compensation Plan ([postmarketOS Change Request 0004](#), "Contributor Compensation Plan"), supplementing volunteer work with paid contribution for otherwise-unsustainable maintenance tasks. Monthly engineering blog updates document sustained, broad activity — kernel work, governance changes, new Trusted Contributors — not just device-count growth ([postmarketOS](#), "in 2026-02: generic kernels"). By a wide margin the strongest community found.
- **Mobian**: volunteer teams, healthy but smaller; no evidence of paid staff.
- **Droidian**: small volunteer team, Open Collective funded at a modest scale; real bus-factor exposure.
- **Mobile NixOS / Maemo Leste**: small contributor bases; Maemo Leste is explicitly recruiting.

5.5 Criterion: device support breadth and depth

- **postmarketOS**: by far the broadest, and the most rigorous about distinguishing "boots" from "actually usable." The official tier docs specify concrete, enforced requirements — e.g. **community** requires a "(close to) mainline kernel," no older than 6 months at time of release ([postmarketOS packaging docs](#), "Device Categorization"). The **main** tier (the highest) is still essentially empty — deliberately: a 2026 change request describes tightening its requirements specifically so that once a device reaches **main** it gets "at least three years of security updates and bug-fix releases," mirroring major distributions' own LTS policies rather than lowering the bar to fill the category ([postmarketOS Change Request 0009](#), "New main device category"). The v26.06 release ("Alpen Avocado," June 2026) lists 254 devices in the testing category alone, spanning exactly the kind of old, abandoned Android hardware this project targets — explicitly naming the Fairphone 4, Pixel 3a, Galaxy S9, Nokia N900, and OnePlus 6 among them ([postmarketOS](#), "v26.06: Alpen Avocado"). Device-count figures vary by source and snapshot date (see Caveats) — treat the **tier definitions** as the reliable signal, not any single headline number.
- **Mobian**: a handful of old-Android phones (OnePlus 6/6T, Poco F1) plus Pine/Librem hardware; thinner per-device documentation, and explicitly defers to postmarketOS's own wiki for mainline boot status rather than maintaining an equivalent of its own.
- **Droidian**: a small curated set (single digits) as of its most recent snapshot releases; broader potential via Halium/GSI but not demonstrated at postmarketOS's scale.

5.6 Criterion: sensor set and GPS completeness

This is dudhya’s core requirement, and postmarketOS’s clearest advantage — directly relevant to this project’s own IIO-sysfs sensor daemon.

Sensors: postmarketOS has the only documented, reusable Qualcomm sensor framework found across every candidate. The architecture (confirmed directly, not assumed): on newer Qualcomm SoCs (SDM845 and similar) “the SoC hypervisor prohibits accessing the sensors directly,” so a dedicated DSP — the Sensor Low Power Island (SLPI), also called Snapdragon Sensor Core (SSC) — must be used instead ([postmarketOS wiki](#), [“Qualcomm Snapdragon Sensor Core”](#)). `libssc` (GLib-based, actively maintained) exposes these sensors to the standard Linux ecosystem via `iio-sensor-proxy` ([libssc](#), [Codeberg](#)). **Directly relevant to this project’s own hardware:** on older Qualcomm SoCs specifically including MSM8974 — the Nexus 5’s own chip — sensors instead share pins with the AP and “these pins could be set to GPIO which allows the AP to access sensors by bit-banging their interfaces,” i.e. closer to dudhya-sensord’s own direct-sysfs approach ([pmaports !4118](#), [“Qualcomm Sensor Manager support”](#)) — confirming, from the OS side, exactly the sensor-access split this project has already lived through hands-on.

GPS: confirmed genuinely achievable, not just theoretically possible, on SDM845 old-Android phones. postmarketOS’s own `gnss-share` project states plainly that Qualcomm modems in devices “such as... OnePlus6... Xiaomi POCOPHONE F1, and many other Qualcomm based devices in postmarketOS provide a QMI Location Interface to retrieve GNSS locations,” via ModemManager, using XTRA A-GPS ([gnss-share issue #10](#)) — the same source already cited in the second-device annex above. First-hand confirmation: “GPS itself works fine” on the Poco F1 ([Drew DeVault](#), [“A review of postmarketOS on the Xiaomi Poco F1”](#)). No equivalent documented GPS path was found for Mobian, Mobile NixOS, or Maemo Leste specifically (they would likely inherit the same ModemManager/QMI mechanism on shared SDM845 hardware, but this was not independently confirmed for those projects).

5.7 Criterion: project health and activity

- **postmarketOS:** very healthy. Regular ~6-month releases, monthly engineering blog updates, active governance evolution, funded contributors — no abandonment risk found.
- **Mobian:** alive, slower cadence, volunteer-only.
- **Droidian:** alive, small team, real bus-factor exposure.
- **Mobile NixOS:** ongoing but low-visibility.
- **Maemo Leste:** active but perpetually pre-“daily driver,” openly recruiting more developers.
- **Ubuntu Touch:** alive but structurally lower tier per above.

5.8 Criterion: developer ergonomics

dudhya’s daemon reads IIO sysfs directly and publishes MQTT. Alpine (musl/Busy-Box) is lightweight and well-documented; **apk** is simple; **pmbootstrap** makes building/flash-ing reproducible; the postmarketOS wiki’s per-device feature-status documentation is unmatched among the candidates researched. Mobian (Debian/apt, glibc) would be more familiar to some tooling and remains a reasonable secondary if a specific future device is better supported there — porting risk is low, since both expose sensors/GPS through the same mainline IIO + ModemManager/QMI plumbing.

5.9 Recommendations

1. **Keep postmarketOS as the primary OS.** It wins on every priority-ordered criterion above; continue investing in the existing infrastructure rather than switching.
2. **Prefer the already-recommended SDM845-generation hardware (OnePlus 6/6T, Poco F1) over the Nexus 5 for GPS-critical work** — this is a hardware-selection reinforcement, not an OS change, and matches the second-device annex above directly.
3. **Keep Mobian noted as a documented fallback, not a switch** — re-evaluate only if a specific future target device turns out to be better supported there.
4. **Continue excluding Halium-based and proprietary options** (Droidian, Ubuntu Touch, Sailfish) — they do not meet this project’s own stated constraints, regardless of raw device counts.

Would change this recommendation: postmarketOS losing its funded contributor model or going quiet for 12+ months; a mandated target device with sensors/GPS working **only** under a Halium-based OS and never achievable on mainline; or Mobian reaching postmarketOS’s own level of tier rigor and old-Android coverage.

5.10 Caveats

- Device-count figures vary meaningfully by source and snapshot date — the v26.06 release notes cite 254 testing-category devices; other sources describe different, later or differently- scoped counts. Treat postmarketOS’s **tier definitions and requirements** as the reliable signal, not any single headline device-count number.
- The live postmarketOS wiki was behind an anti-bot challenge at research time; several claims rely on GitLab issues/merge requests, official packaging docs, and cached search snippets rather than a live-rendered wiki page — re-verify the actual device feature tables before finalizing any specific hardware choice.

- GPS via SUPL (as opposed to XTRA A-GPS) is itself flagged as unconfirmed on the postmarketOS wiki's own troubleshooting page — XTRA is the confirmed-working path, not SUPL.
- No equivalent GPS/sensor documentation was found (or ruled out) for Mobian, Mobile NixOS, or Maemo Leste specifically — absence of findings, not a claim that they lack the capability.
- Community-strength assessment here is qualitative (governance structure, funding, release cadence, blog activity) rather than exact commit-count analytics, which could strengthen a future revision of this comparison.