Lucid Ownership Experience

Coding Solution Drafts for Identified Gaps

This document lists assumptions first, then provides draft coding solutions for the 16 ownership-experience gaps identified from real-world use of a Lucid Air. Solutions are framed for an Android Automotive OS vehicle stack with a vehicle-property / CAN data layer, dual-screen SystemUI, and an existing driver-profile system. They are intended as practical starting points for engineering discussion, not production-ready code.

Assumptions

1. Vehicle software is based on Android Automotive OS with access to a vehicle property/API layer (or equivalent) that can expose real-time power, energy, SoC, odometer, trip, and climate data. Efficiency metrics (mi/kWh, projections) can be derived from existing CAN/vehicle signals or a new telemetry service.

2. UI modifications target the dual-screen stack (Cockpit Panel + Pilot Panel) and can be implemented via custom SystemUI / launcher / media / navigation apps without a full OS rewrite.

3. Ambient light sensors, magnetometer/IMU (or GPS-derived heading), and camera feeds are available to the infotainment domain (with appropriate permissions and safety isolation).

4. Driver profiles already store persistent settings; extending them for high beams, seat positions, and HAL preferences is feasible.

5. DreamDrive Pro hands-on detection currently monitors certain inputs; adding steering-wheel button events is possible via the existing input/ADAS interface (subject to functional safety review).

6. Dashcam/recording is constrained by NVIDIA SoC architecture, storage, thermal, and privacy/regulatory limits; a best-effort continuous or event-triggered recorder is assumed possible for the Pilot Panel or a background service.

7. Mobile app changes assume the existing Lucid cloud/API surface; confirmation dialogs are client-side and can be made configurable or removed for low-risk actions.

8. Older hardware (pre-UX 3.0 / pre-second-gen processor) may require feature gating or the paid upgrade path already discussed.

9. All changes respect existing safety constraints (e.g., no profile switching that drops critical nav while moving, no UI that increases distraction beyond current design).

Coding Solution Drafts

1–2. HAL Button + Richer Left-Screen Efficiency Display

Create a new EfficiencyService that aggregates power (kW), energy (kWh), distance, and SoC into rolling mi/kWh, trip projections, and trend buffers. Expose it via a vehicle property or binder. Replace the Themes button slot with a HAL tile that launches a full-screen or split view (live mi/kWh, projected range, consumption graph, trip summary). Move Themes under Settings → Display. On the left Cockpit Panel, add a compact efficiency card (live mi/kWh + score badge + remaining trip energy).

```
// Pseudocode – EfficiencyService (Kotlin)
class EfficiencyService : Service() {
```

```
    private val powerKw = vehicleProperty.get(POWER_KW)
    private val energyKwh = vehicleProperty.get(ENERGY_KWH)
    private val distanceMi = vehicleProperty.get(ODOMETER_MI)

    val liveMiPerKwh: Flow<Float> = combine(powerKw, energyKwh, distanceMi) { p, e, d ->
        if (e > 0) d / e else 0f
    }.stateIn(...)

    fun projection(soC: Float, bufferPercent: Float): Float =
        remainingEnergy * efficiency * (1 - buffer)
}
```

3. Power Meter Readability (DreamDrive Pro)

In the ADAS overlay or instrument cluster renderer, change the power bar draw call from gray-on-gray to green fill on gray background. Keep the existing white regen indicator. Threshold coloring optional (e.g., green < 50% power, amber above).

```
// In PowerMeterView
paint.color = if (powerRatio > 0) Color.GREEN else Color.WHITE
canvas.drawRect(barRect, paint)
```

4. Compass on Nav

Subscribe to magnetometer or GPS heading in the navigation process. Overlay a small compass rose or numeric heading (e.g., “NW 315°”) in a non-critical corner of the nav map view. Update at 1–2 Hz; hide when map is zoomed out if clutter is a concern.

```
// Nav overlay subscription
headingFlow.collect { heading ->
    compassView.text = formatHeading(heading)  // e.g. "NW 315°"
    compassView.visibility = if (zoomLevel > MIN_ZOOM) VISIBLE else GONE
}
```

5. Driver-Configurable Arrival Buffer in Route Planning

Add a user setting (default 10–15%, range 0–40%) stored in profile or global prefs. In the routing engine, subtract the buffer from usable SoC before computing stops. Surface the control in the route options panel and show the resulting arrival SoC estimate.

```
fun planRoute(destination: LatLng, bufferPercent: Float) {
    val usableSoC = currentSoC * (1 - bufferPercent / 100f)
    router.plan(destination, usableSoC, preferredNetworks)
}
```

6. Auto-Dim Based on Ambient Light

Register for ambient light sensor events. Map lux to brightness percentage (with hysteresis to avoid flicker). Apply to both screens via WindowManager or display brightness API. Allow override in Settings and tie to profile if desired.

```
sensorManager.registerListener(ambientListener, SENSOR_LIGHT, SENSOR_DELAY_NORMAL)
fun onLux(lux: Float) {
    val target = mapLuxToBrightness(lux)  // with hysteresis band
    displayManager.setBrightness(target)
}
```

7. Right Home Screen Rework

Deduplicate time/temperature. Replace the redundant widgets with a consolidated card: next turn + ETA, current efficiency, or media now-playing. Keep the existing card-swipe model from UX 3.0; just change the data source for that slot.

8. Burn-in Mitigation on Left Control Screen

In the icon layout pass, apply a small periodic offset (±10 px) to static icons. Use a low-frequency timer (every 30–60 s) or frame-based jitter within the existing hit-box so touch targets remain stable. Reset on interaction.

```
iconView.translationX = (System.currentTimeMillis() / 30_000 % 3 - 1) * 10f
// Reset on touch: iconView.translationX = 0f
```

9. Shrink Preconditioning UI

Remove the full-screen takeover. Show a small persistent icon or short label (“Battery Conditioning”) above the drive-mode selector (Smooth/Swift/Sprint) on the Pilot Panel. Tap expands a compact status sheet. State driven by the same climate/precondition service.

10. Numbered Volume Steps

Expose volume as 1–20 (or 0–20) in the media volume controller. Display the numeric value next to the slider or in the quick controls. Map to the underlying audio HAL linearly.

```
volumeLabel.text = "${(normalizedVolume * 20).toInt()}/20"
audioManager.setStreamVolume(STREAM_MUSIC, mappedStep, 0)
```

11. Count Steering-Wheel Inputs for Hands-on Check

In the DreamDrive input monitor, subscribe to steering-wheel button events (next track, previous, volume up/down, turn signal). Any of these increments the “engaged driver” counter the same way a torque or touch event does. Keep the existing timeout and escalation logic.

```
wheelButtonEvents.onEach { event ->
    handsOnDetector.registerEngagement(source = "wheel:${event.action}")
}.launchIn(scope)
```

12. Spotify Integration Refresh

Update the media session / Spotify Android Automotive integration (or the custom Lucid media app) to use current Spotify SDK patterns: better browse hierarchy, search, queue management, and album art loading. Reduce multi-tap paths; cache recent playlists. This is primarily a media-app rewrite rather than a vehicle-core change.

13. Auto High Beams Tied to Driver Profile

Add a boolean autoHighBeamsEnabled to the driver profile schema. On profile load, write the value to the lighting vehicle property. Persist changes made while that profile is active.

```
profile.onLoad { p ->
    vehicleProperty.set(LIGHTING_AUTO_HIGH_BEAMS, p.autoHighBeamsEnabled)
}
```

14. Seat Symmetry Button

Add a one-tap control (in seat settings or a quick action) that copies the driver's seat position (fore-aft, height, recline, lumbar) to the passenger seat motors. Store as an optional profile action or one-shot command.

```
fun mirrorSeats() {
    val driverPos = seatController.getPosition(SEAT_DRIVER)
    seatController.setPosition(SEAT_PASSENGER, driverPos)
}
```

15. Dashcam Attempt

Background service that records from the forward (and optionally cabin) camera when the vehicle is in Drive or when a trigger occurs (hard brake, button press). Write to local storage with a circular buffer (e.g., last 30–60 min). Provide a simple playback UI on the Pilot Panel. Gate behind a setting; note NVIDIA/storage/thermal limits and privacy prompts. Start with event-triggered or Drive-only to stay within practical constraints.

```
class DashcamService : Service() {
    fun startRecording() {
        camera.open(FORWARD_CAMERA).recordTo(
            circularBuffer(maxDuration = 60.minutes, storageDir)
        )
    }
}
```

16. Lucid App — Remove Redundant Confirmations

In the mobile client, for low-risk actions already initiated by the user (lock/unlock, climate on/off, charge start/stop, frunk/trunk), skip the confirmation dialog or make it optional via a "Don't ask again" preference. Keep confirmations for irreversible or high-impact actions. Update the API call path to fire the command immediately after the initial tap when the preference is set.

```
if (prefs.skipConfirmFor(action) || !action.isHighRisk) {
    api.execute(action)
} else {
    showConfirmDialog(action)
}
```

Implementation Priority Note

Priority order for a first implementation wave would typically be: efficiency display/HAL (1–2), power meter (3), arrival buffer (5), auto-dim (6), preconditioning shrink (9), and app confirmation cleanup (16). These improve daily ownership with relatively contained scope and leverage existing vehicle properties and UI frameworks.

These drafts assume the existing Android Automotive + vehicle-property architecture and focus on the highest-leverage changes for the listed gaps. They are starting points for engineering review and can be expanded into full designs, UI mocks, or refined implementation plans as needed.