F1 25 UDP Telemetry – Data Ingestion & JSON Stream Spec

1. Scope (What this project does)

This project **only** handles telemetry data ingestion, decoding, normalization, and export from **F1 25** via UDP.

In scope - Receive F1 25 UDP packets - Decode official 2025 packet structures - Merge packets into a live state model - Select nearby cars - Output a readable **JSON stream**

Explicitly out of scope - Racing AI - Coaching logic - Overtake/defend strategy - Decision-making or ML

---

2. Telemetry Configuration (Locked)

- **UDP format:** 2025
- **UDP send rate:** 30 Hz
- **UDP port:** 20777
- **Platform:** Windows
- **Modes supported:**
- Online multiplayer
- Offline / AI races

---

3. Packet Handling Strategy

3.1 Supported packet types

All official F1 25 packets may be received, with emphasis on: - Motion - Session - Lap Data - Participants - Car Telemetry - Car Status - Car Damage

Other packets are decoded and stored if present but not required for output.

3.2 Version & robustness rules

- Expect **2025 packet layout**
- Perform **sanity checks** on:
- Packet size
- Header fields
- Field value ranges
- Do **not hard-fail** on minor mismatches

3.3 Known spec issue handling

- `LapData.m_gridPosition` and `LapData.m_driverStatus` may be swapped in-game
- Detect via range validation
- Apply workaround **only if detected**, with one-time warning

3.4 Endianness

- Detect automatically via header and field plausibility checks

---

4. State Model

Because packets arrive at different rates, maintain a live merged state: - Player car state - Per-car state for all 22 cars - Session / track metadata - Latest frame & session timestamps

All JSON output frames are produced from this state store.

---

5. Nearby Cars Selection (Final Rules)

5.1 Time-gap computation

For each car: - Compute delta-to-leader (seconds) from LapData - Player index from packet header - `gapToPlayerSec = |deltaLeaderSec(car) - deltaLeaderSec(player)|`

5.2 Eligibility

- Exclude player car
- Exclude inactive / invalid cars
- Keep cars with `gapToPlayerSec ≤ 1.5s`

5.3 Sorting & limits

- Sort by `gapToPlayerSec` ascending
- Select **up to 6 cars**

5.4 Tie-breaking & bias

- Prefer cars **ahead of the player**
- Target composition: **~4 in front / ~2 behind** when possible
- If insufficient cars on one side, fill from the other

---

6. Tyre Wear & Degradation

- Use **Car Damage packet** for tyre wear (% per wheel)
- Online multiplayer may restrict opponent telemetry
- If unavailable, output `null` (never misleading zeros)

---

7. Track & Position Model

7.1 Position reference

- Primary: `LapData.m_lapDistance` (meters)
- Secondary: world position `(X,Y,Z)` for spatial context

7.2 Track map approach

- **Hybrid**:
- Auto-build track maps from telemetry laps
- Allow replacement with curated maps later

Upcoming corner logic is supported by the data model but **not implemented in this phase**.

---

8. JSON Output Stream

8.1 Format

- Newline-delimited JSON (**JSONL / NDJSON**)
- One frame per line
- Emitted at **30 Hz**

8.2 Frame characteristics

Each frame is a merged snapshot containing: - Player inputs: steering, throttle, brake, clutch, gear, speed - Player race position and lap distance - Nearby cars list (per rules above) - Tyre wear where available - Session metadata (trackId, sessionType, etc.)

8.3 Data availability rules

- Fields not available due to privacy or packet absence → `null`
- Latest-known values reused until updated by a new packet

---

9. Logging (Optional)

- Raw UDP packet logging (binary)

- Decoded frame logging (JSONL)
- Not required for first implementation

---

10. Deliverable

A Windows application that: - Receives F1 25 UDP telemetry on port 20777 - Decodes packets reliably - Maintains a live state model - Outputs a clean, stable **JSON telemetry stream** at 30 Hz

No racing intelligence, coaching, or AI behavior is included in this phase.