AI-Positioning PoC: Complete Technical & Organizational Guide

Document Version: 3.1 (Final)

Last Updated: January 19, 2026

Status: Final Reference Document (Intent-Driven AI-Native PPaaS)

Audience: Technical teams (beginner to intermediate level)

Executive Summary

This document provides a complete reference for the **AI-Positioning Proof-of-Concept (PoC)**, an **intent-driven AI-native precise positioning-as-a-service (PPaaS)** system delivered over ATSC 3.0 broadcast.

The PoC demonstrates how **high-level operator or service-provider intents** (e.g., "maximize accuracy", "maximize reliability in urban canyons", "optimize ATSC spectrum usage") are automatically translated by an AI agent into **real-time configuration of the entire positioning pipeline**:

- ATSC 3.0 PLP parameters (bandwidth, FEC, modulation, redundancy)
- RTK vs SSR correction mix and update rates
- Bitmap/grid tile resolution and refresh rate
- Edge vs UE fusion parameters (GNSS + IMU + broadcast corrections)

The system integrates:

- **ATSC 3.0 broadcast delivery** of GNSS corrections, maps, and bitmap-based error grids
- **RTK-style high-precision corrections** (RTCM/SSR) via broadcast, with optional unicast fallback
- **AI-driven adaptive control** of correction rate, message type, edge compute, and bandwidth allocation

Primary KPIs:

- **Positioning accuracy:** 3–10 cm horizontal / vertical where feasible
- **Reliability:** High valid-correction availability, including rural and urban canyon conditions
- **Correction delivery latency:** Broadcast → UE decode within acceptable RTK bounds
- **ATSC 3.0 spectrum efficiency:** Bits/s/Hz used for positioning PLPs
- **Service continuity:** Dropout rate across rural and urban canyon environments

Key Innovation

1. **Traditional vs AI-Native Behavior Demonstration**

   The PoC explicitly contrasts:

   - **Traditional System (Baseline):**
     - Static RTK correction configuration
     - Fixed ATSC parameters (redundancy, FEC, PLP mapping, tile resolution)
     - No intent awareness, no feedback loop
   - **AI-Native Precise Positioning-as-a-Service:**
     - Operator provides **intents** ("Guarantee sub-10 cm accuracy", "Optimize ATSC spectrum use", "Maximize reliability in urban canyon")
     - AI agent continuously ingests telemetry and network conditions
     - AI outputs real-time configuration of broadcast, correction, and fusion parameters

   This comparison is demonstrated across **three concrete PoC test scenarios**:

   1. **Scenario 1:** High-Accuracy Drone in Rural Area
   2. **Scenario 2:** Low-Bandwidth Rural ATSC Coverage
   3. **Scenario 3:** Urban Canyon Environment

2. **Team Structure**

- **Team 1 – GNSS Positioning Engineer**
- **Team 2 – Broadcast Systems Engineer**
- **Team 3 – AI/ML Systems Engineer**

3. **Project Plan**

- Duration: **12 weeks**, 4 phases (Foundation → Core Modules → Integration & AI → Production & Demo)
- Tools: **100% open-source**, zero license fees
- Deliverable: **Production-ready PoC** suitable for deployment on **Qualcomm AI Hub**, including:
  - Traditional baseline vs AI-native behavior
  - Demo assets for all three scenarios
  - End-to-end narratives and **three canonical test scenarios for PoC evaluation**

PART 1: UNDERSTANDING THE PROBLEM STATEMENT

1.1 The Core Problem We're Solving

Real-World Scenario: Why This Matters

Imagine an autonomous vehicle or drone operating across mixed environments:

- **In open sky (highway or rural field):** GNSS works well → ±1.5 cm RTK accuracy
- **In urban canyons:** Signals suffer from multipath and partial blockage → errors jump to tens of centimeters or meters
- **In tunnels / deep blockage:** GNSS can disappear entirely → system must rely on dead-reckoning and map constraints

The Challenge:

- Autonomous vehicles, drones, and robots need **centimeter-level precision** for lane-keeping, obstacle avoidance, precise landings, and geo-fencing.
- **GNSS alone** cannot guarantee this in **urban canyons, rural fringe coverage, or blockage zones**.
- Existing solutions rely on:
  - Expensive LiDAR (>$50K)
  - High-grade inertial systems (>$30K)
  - Cellular-based correction delivery (NTRIP) with coverage/cost issues

Our Solution:

A **broadcast-based, AI-orchestrated positioning service** that:

1. Uses a **reference station network** to measure GNSS errors (RTK/SSR/bitmap-based models).
2. Encodes corrections in **RTCM** and/or compact grid formats.
3. **Broadcasts** them over **ATSC 3.0** PLPs (tens of kilometers range, mass reach).
4. Uses an **AI agent** to adapt broadcast and positioning parameters to **operator intent + real-time conditions**.
5. Helps vehicles and drones maintain **3–10 cm accuracy**, with robust reliability, while **optimizing ATSC spectrum usage**.

1.2 Current State of GPS / GNSS Technology

Standard GNSS (What Everyone Uses)

```
Accuracy:       ±5–10 meters
Why Limited:    Code-phase measurements only
Use Case:       "You are somewhere in this city block"
Problem:        Not enough for autonomy, precision landing, or lane-level services
```

RTK GNSS (Real-Time Kinematic – Our Accuracy Target)

```
Accuracy:       ±1.5–2 cm (centimeter-level)
How:            Uses carrier-phase measurements + corrections from reference stations
Range:          ~20–50 km from base (depending on ionosphere/troposphere)
Problem:        Requires low-latency, reliable correction delivery channel
Our Twist:      Deliver corrections over ATSC 3.0 (broadcast PLPs) + AI control
```

Why ATSC 3.0 for Broadcasting?

```
┌─────────────────────────────────────────────────────────────┐
│              ATSC 3.0 (Next-Gen Digital TV)                 │
├─────────────────────────────────────────────────────────────┤
│ Broadcast Range:    30–50 km (metropolitan-scale)           │
│ Bandwidth:          6 MHz                                   │
│ Spectrum:           Already licensed for TV                 │
│ Data Capacity:      5–57 Mbps depending on robustness       │
│ Latency:            ~5–10 seconds end-to-end                │
│                                                           │
│ Compare to:                                               │
│ • Cellular (LTE/5G): Per-UE cost, spectrum constraints    │
│ • Wi-Fi: Short range, infrastructure heavy                │
│ • Satellite: Higher latency, limited urban penetration    │
│                                                           │
│ Why ATSC 3.0?                                             │
│ • One-to-many delivery (one transmitter → thousands UEs)  │
│ • Robust mobile reception (200+ km/h)                     │
│ • Tunable robustness via PLPs and FEC                     │
└─────────────────────────────────────────────────────────────┘
```

1.3 Why AI & Intents Are Needed (The Smart Part)

Without AI: Static, "Traditional" Delivery

Traditional approach:

```
Operator: "Use RTK corrections over broadcast"
System:
  • Fixed correction rate (e.g., 1 Hz)
  • Fixed FEC rate (e.g., 15%)
  • Fixed PLP mapping
  • Fixed bitmap/grid resolution
  • No awareness of:
      - Rural vs urban canyon
      - Drone vs car vs stationary receiver
      - Spectrum pressure
      - Operator business goals
```

Problems:

- **No intent awareness:** Cannot express "Today I care about sub-10 cm accuracy for drones" vs "Now I care more about saving spectrum."
- **Inefficient spectrum use:** Same bitrate and redundancy everywhere.
- **No feedback loop:** Broadcast doesn't "see" if receivers achieve FIX or not.
- **Rigid behavior:** Works "OK" in some conditions, fails badly in edge cases.

With AI + Intents: Intelligent, Adaptive PPaaS

```
Operator sets intent:
  "Guarantee sub-10 cm accuracy for rural drone corridor"
  or
  "Optimize ATSC spectrum use for coverage campaign"
  or
  "Maximize reliability in this urban canyon zone"

AI Orchestrator:
  • Observes:
      - Fleet RTK modes (FIX/FLOAT/STAND-ALONE)
      - Accuracy residuals
      - PLP performance (SNR, packet loss, FEC performance)
      - Environment (rural/open sky vs urban canyon)
  • Decides:
      - Correction update rate (1–10 Hz)
      - RTK vs SSR mix
      - Bitmap tile resolution and frequency
      - PLP FEC and modulation
      - Redundancy / repetition patterns
  • Enforces:
      - Real-time ATSC + positioning configuration
      - Edge and UE fusion policies (GNSS+IMU+bitmap)
```

Benefits:

- **Intent-driven operation:** Operator/business goals drive configuration, not hard-coded profiles.
- **Adaptive behavior:** Broadcast and positioning adapt to actual conditions and fleet performance.
- **Traditional vs AI-native comparison:** PoC explicitly showcases how AI-native orchestration outperforms a static baseline in three scenarios.

1.4 Key Problem Statements

Problem 1: Precision in Sparse Rural Environments

- **What:** Drones or vehicles in rural corridors need **sub-10 cm accuracy** for delivery, inspection, or agriculture.
- **Challenge:** GNSS geometry can be good, but backhaul or per-UE bandwidth is limited and unicast correction delivery doesn't scale.
- **Our Approach:** Broadcast RTK/bitmap corrections with AI-controlled rate and robustness, tuned by **"Guarantee sub-10 cm accuracy"** intent.

Problem 2: ATSC Spectrum Efficiency

- **What:** Broadcasters and operators must **protect ATSC spectrum usage**, especially when positioning is one of multiple services.
- **Challenge:** Over-provisioned corrections waste capacity that could serve other data/services.
- **Our Approach:** Intent **"Optimize ATSC spectrum use"** drives the AI to reduce bitrate, adjust SSR vs RTK, and use lower-rate encodings while maintaining adequate accuracy.

Problem 3: Reliability in Urban Canyons

- **What:** Dense urban environments cause multipath, partial blockage, and geometry degradation, threatening reliability.
- **Challenge:** Fixed configurations either:
  - Over-spend spectrum everywhere to cover urban worst cases, or
  - Underperform in urban canyons.
- **Our Approach:** Intent **"Maximize reliability"** in specific zones causes AI to:
  - Strengthen error grid resolution
  - Increase integrity updates
  - Adjust multipath mitigation and fusion weights.

Problem 4: Lack of Feedback Loop (Traditional)

- **What:** Traditional systems don't receive systematic telemetry from receivers.
- **Impact:** No way to verify if broadcast choices work; no continuous improvement.
- **Our Approach:** Telemetry from UEs feeds the AI Orchestrator, which continuously refines configuration.

PART 2: TECHNICAL DEEP DIVE – COMPLETE ARCHITECTURE

2.1 Intent-Driven PPaaS System Overview

At a high level, the PoC is structured as **five logical layers**:

1. **Reference Station Network (GNSS Ground Segment)**

   - Multi-GNSS receivers (GPS, Galileo, BeiDou, etc.)
   - Produce:
     - RTK corrections (RTCM/MSM)
     - Bitmap/grid-based error models (ionosphere, troposphere, multipath probability)

        - SSR data for wide-area fallback
    - Feed data with precise timestamps into the **Broadcast Core**.

2. **Broadcast Core (AI-Native Positioning Orchestrator Center)**
    - Hosts the **Intent-Driven AI-Native Positioning Orchestrator**.
    - Responsibilities:
        - Interpret high-level **operator intents** (accuracy, reliability, spectrum efficiency).
        - Select and configure ATSC PLPs for positioning content.
        - Choose optimal RTK vs SSR mix.
        - Adjust bitmap tile resolution and update frequency.
        - Control redundancy, FEC, and repetition.
        - Instruct **Broadcast Edge** processing policies.
3. **Broadcast RAN (ATSC 3.0 Transmitter)**
    - Implements PLPs such as:
        - **PLP-A:** RTK correction stream
        - **PLP-B:** Bitmap/grid-based error maps + integrity
        - **Optional PLP-C:** Specialized low-power or extended coverage services
    - Applies OFDM modulation, FEC, and physical layer configurations.
4. **Broadcast Edge**
    - Optional component near receivers (e.g., edge servers, roadside units).
    - Performs:
        - Local multipath prediction
        - Correction smoothing / gap filling
        - Tile caching
        - UE-specific fusion policy hints
    - Receives configuration from the AI Orchestrator.
5. **UE / Receiver Layer (GNSS + ATSC Device)**
    - Dual receiver:
        - GNSS chipset capable of RTK precision.
        - ATSC 3.0 demodulator for PLP reception.
    - Software stack:
        - Bitmap/grid decoder.
        - RTK/SSR correction consumer.
        - AI-assisted fusion engine combining GNSS, IMU, and broadcast corrections.
    - Produces high-accuracy positions, even in weak-signal or high-mobility conditions.

2.2 Detailed System Architecture

COMPONENT 1: Base Station & Reference System

- Multi-constellation GNSS receiver at a **known position** (surveyed to mm accuracy).
- Continuously computes **error vectors** (ΔX, ΔY, ΔZ) and atmospheric/clock corrections.
- Outputs:
    - Per-satellite error vectors
    - Atmospheric delay models
    - Clock offsets
    - Signal quality metrics

COMPONENT 2: RTCM Correction Generator

- Encodes raw error vectors into **RTCM 3.x** binary frames (100–300 bytes).
- Supports message types:
    - 1004, 1005, 1012, etc.
- Output rate and composition (RTK vs SSR) become **AI-controllable knobs** per intent.

COMPONENT 3: Coverage / Bitmap Map Generator

- Generates **100×100 pixel tiles** representing:
    - Good coverage (white)
    - Blocked/poor coverage (black)
    - Degraded multipath-prone zones (gray)
- Tiles may represent:
    - Rural corridors
    - Urban canyon blocks
    - Drone corridors
- Tile resolution and update rate are **AI-adjustable** by intent.

COMPONENT 4: Data Aggregator & AI Feedback Controller

(**Intent-Driven Positioning Orchestrator**)

Inputs:

- **Intents:**
    - "Guarantee sub-10 cm accuracy" (Scenario 1)
    - "Optimize ATSC spectrum use" (Scenario 2)
    - "Maximize reliability in urban canyon" (Scenario 3)
- **Network & Environment State:**
    - ATSC PLP load, SNR, FEC performance
    - GNSS satellite visibility and geometry
    - Urban density/multipath intensity
    - UE velocity distributions
    - Latency of reference station feeds

- **Quality Metrics (from UEs):**
  - RTK mode (STAND-ALONE / FLOAT / FIX)
  - Horizontal / vertical residuals
  - Correction age
  - Integrity indicators
  - PLP packet loss, bit error rates

Outputs (AI-Controlled Parameters):

1. **ATSC 3.0 Broadcast Parameters**

   - PLP bandwidth and FEC code rate
   - Modulation scheme per PLP (QPSK vs 16-QAM vs 256-QAM, etc.)
   - Correction update rate (e.g., 0.5–10 Hz)
   - RTK vs SSR vs bitmap mix
   - Redundancy / repetition patterns
   - Tile resolution (coarse vs fine grids) and refresh rate

2. **Positioning Algorithm / Fusion Parameters**

   - Weights for GNSS vs IMU vs broadcast corrections
   - Multipath mitigation thresholds (especially in urban canyons)
   - Station/SSR source selection
   - Edge vs UE processing split

COMPONENT 5: ATSC 3.0 Broadcast Encoder & RAN

Key mechanisms:

- ALP packetization of RTCM and tiles.
- FEC using LDPC/Reed-Solomon with AI-controlled overhead.
- OFDM modulation with configurable subcarriers, guard intervals, and modulation schemes.
- Multiple PLPs:
  - PLP-A: Highly robust corrections for mobile receivers.
  - PLP-B: Maps/tiles and integrity information.
  - PLP-C (optional): Experimental or extended coverage services.

COMPONENT 6: Vehicle / Drone Receiver & RTK Processing

- May operate in **traditional (fixed) mode** or **AI-orchestrated mode** for PoC comparison.
- Telemetry is explicitly designed to **feed back** KPIs for scenario evaluation.

2.3 Data Formats & Communication Protocols

RTCM Frame Format (Binary)

```
┌───────┬───────┬───────┬───────┬────────┬───────┐
│ PRE   │ RSV   │ LEN   │ TYPE  │PAYLOAD│ CRC   │
├───────┼───────┼───────┼───────┼────────┼───────┤
│ 1B    │ 6b    │ 10b   │ 12b   │ Var    │ 3B    │
└───────┴───────┴───────┴───────┴────────┴───────┘

Preamble (0xD3):    Synchronization marker
Reserved:           Future expansion
Length:             Size of message in bytes
Type:               Message type (1004, 1005, 1012, etc.)
Payload:            Actual correction data
CRC-24:             Error detection

Common Message Types:
  1004 = RTK Base Station Observations
  1005 = Base Station Coordinates
  1012 = GLONASS Observations
```

Vehicle Telemetry JSON Format

```
{
  "timestamp": 1705094400,
  "vehicle_id": "vehicle_001",
  "location": {
    "latitude": 37.580746,
    "longitude": 126.892210,
    "height_m": 106.950
  },
  "rtk_metrics": {
    "mode": "FIX",
    "position_error_cm": 1.5,
    "num_satellites_used": 12,
    "signal_strength_db_hz": 39.8,
    "convergence_time_sec": 18.3
  },
  "environment": {
    "urban_density": 0.2,
    "is_in_tunnel": false,
    "is_in_canyon": false,
    "predicted_blockage_ahead": false
  },
  "vehicle_state": {
    "speed_kmh": 80,
    "heading_deg": 45,
    "confidence": 0.98
  }
}
```

AI Broadcast Command JSON Format

```
{
  "timestamp": 1705094500,
  "intent": "maximize_reliability",
  "broadcast_config": {
    "redundancy": 1.8,
    "tile_resolution": "high",
    "update_frequency_hz": 3.0,
    "plp_mode": "mobile",
    "fec_overhead_pct": 30
  },
  "reasoning": {
    "current_fix_pct": 60,
    "avg_convergence_sec": 40,
    "urban_canyon": true,
    "decision": "INCREASE_ROBUSTNESS_FOR_URBAN_CANYON"
  },
  "confidence": 0.91
}
```

PART 3: THREE CANONICAL TEST SCENARIOS

The PoC demonstrates traditional vs AI-native behavior across three distinct scenarios aligned with the original problem statement.

3.1 Scenario 1: High-Accuracy Drone in Rural Area

Intent: "Provide < 3 cm accuracy"

Environment & Use Case

- Rural or ex-urban corridor (e.g., inspection, agriculture, logistics).
- Good GNSS visibility, minimal multipath.

- ATSC coverage is generally good.
- Drones require **very tight accuracy** for:
    - Landing on pads
    - Following narrow corridors
    - Infrastructure inspection

Traditional System Behavior (Baseline)

- Static configuration:
    - Fixed RTK correction rate (e.g., 1 Hz)
    - Medium FEC and redundancy
    - Medium-resolution tiles
- No awareness of:
    - Drone vs car vs stationary UE
    - Operator's specific accuracy demand
- Result:
    - Accuracy can be good, but not **guaranteed** or **prioritized**

AI-Native PPaaS Behavior

- Intent **"Provide < 3 cm accuracy"** is set.

- AI actions:

    - **Increase correction rate** (within allowed budget)
    - **Use high-resolution bitmap tiles** where needed (drone corridor)
    - **Boost PLP reliability via stronger FEC** and redundancy for the relevant PLP

- Expected effects:

    - Higher consistency in achieving sub-3 cm accuracy
    - Possibly higher bits/s/Hz in the drone corridor PLP, but only when needed and intent-justified

---

3.2 Scenario 2: Low-Bandwidth Rural ATSC Coverage

Intent: "Minimize ATSC spectrum usage"

Environment & Use Case

- Large rural area where ATSC transmitter must cover:
    - Positioning services
    - Other datacast or broadcast services

- Spectrum is precious; per-UE or per-service overprovisioning is undesirable
- Tight capacity budget for positioning PLPs

Traditional System Behavior (Baseline)

- Static configuration:
  - Moderately robust RTK stream at fixed bitrate
  - Conservative FEC settings that remain on at all times
- Result:
  - Spectrum used even when few UEs need high precision
  - No dynamic reduction in bitrate when conditions are good

AI-Native PPaaS Behavior

- Intent **"Minimize ATSC spectrum usage"** is set.

- AI actions:

  - **Decrease correction bitrate** where possible:
    - Switch to lower-rate **SSR/bitmap encoding** for suitable receivers
    - Reduce RTK frequency in stable periods
  - **Reduce PLP bandwidth** where UEs can tolerate slightly slower convergence

- Expected effects:

  - Lower bits/s/Hz usage for positioning PLPs vs baseline
  - Accuracy may be slightly reduced but remains within agreed service levels
  - Clear demonstration of **spectrum-efficiency tuning** driven by intent

---

3.3 Scenario 3: Urban Canyon Environment

Intent: "Guarantee service continuity in weak-signal regions"

Environment & Use Case

- Dense urban area with tall buildings ("urban canyon")
- Partial blockage, strong multipath, and challenging GNSS geometry
- Target is to **avoid outages** and **keep position usable** for autonomy and navigation

Traditional System Behavior (Baseline)

- Static configuration:

    - Single moderate RTK rate, fixed tiles, moderate FEC
- Result:
    - May underperform during the worst urban canyon segments
    - To be safe everywhere, one would need to **overprovision** across the entire area → wasteful

AI-Native PPaaS Behavior

- Intent **"Guarantee service continuity in weak-signal regions"** is set.
- AI actions:
    - **Strengthen error grid resolution:**
        - Use finer tiles for multipath-prone blocks
    - **Increase integrity update frequency:**
        - Allow UEs to detect and reject bad measurements more effectively
    - **Adjust multipath prediction and fusion weighting:**
        - Downweight suspect GNSS signals
        - Upweight IMU and map constraints
    - Increase PLP robustness (higher FEC, lower-order modulation) **for the canyon area**, not everywhere
- Expected effects:
    - Reduced frequency/duration of RTK outages vs traditional
    - Better continuity of usable positioning in the canyon area
    - Acceptable global spectrum cost because enhancement is **localized and intent-driven**

PART 4: DETAILED TEAM ORGANIZATION & WORK DISTRIBUTION

4.1 Three-Person Team Structure

```
┌─────────────────────────────────────────────────────────────────────┐
│     3-PERSON TEAM ORGANIZATION                                      │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│  ┌─────────────────┐  ┌─────────────────┐  ┌──────────────┐       │
│  │ TEAM 1: GNSS    │  │ TEAM 2:BROADCAST│  │ TEAM 3: AI   │       │
│  │ Positioning     │  │ Systems         │  │ /ML          │       │
│  │ Engineer        │  │ Engineer        │  │ Engineer     │       │
│  │                 │  │                 │  │              │       │
│  │ Domain:         │  │ Domain:         │  │ Domain:      │       │
│  │ • GPS/GNSS      │  │ • RF/Telecom    │  │ • AI/ML      │       │
│  │ • RTK           │  │ • ATSC 3.0      │  │ • Neural     │       │
│  │ • Positioning   │  │ • FEC/Encoding  │  │   Networks   │       │
│  │ • RTKLIB        │  │ • Signal proc   │  │ • PyTorch    │       │
│  │                 │  │ • Modulation    │  │ • Intent     │       │
│  │                 │  │ • Intent        │  │   Parser     │       │
│  │                 │  │   Handling      │  │              │       │
│  └─────────────────┘  └─────────────────┘  └──────────────┘       │
│            │                   │                       │            │
│            └───────────────────┴───────────────────────┘            │
│                         │                                           │
│                    SYNC POINTS:                                     │
│                   • Weekly 1-hour meetings                          │
│                   • Data handoff between modules                    │
│                   • Integration tests every 2 weeks                 │
│                   • Phase completion gates                          │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

4.2 TEAM 1: GNSS Positioning Engineer - ANIRUDH

Responsibilities

PHASE 0 (Weeks 1–2): FOUNDATION & SETUP

- Install RTKLIB, verify compilation
- Run baseline RTK on sample data, measure "no broadcast" errors
- Tag baseline performance for three scenarios

PHASE 1 (Weeks 3–6): CORE GNSS & SCENARIO DATA

- RTCM generator
- Coverage map generator
- Scenario simulator
- **Explicit Scenario Support for the Three PoC Cases:**
    1. **Scenario 1 – High-Accuracy Drone in Rural Area**
        - Generate trajectories and raw GNSS data representing drone flight corridors
        - Baseline traditional RTK results + AI-mode-friendly datasets
    2. **Scenario 2 – Low-Bandwidth Rural ATSC Coverage**
        - Model lower SNR / marginal ATSC coverage
        - GNSS logs where corrections might have to be sparser
    3. **Scenario 3 – Urban Canyon Environment**
        - Generate or synthesize data with strong multipath and partial blockage
        - Residuals and mode transitions (FIX → FLOAT → STAND-ALONE)
- Deliverables:
    - `rtcm_generator.py`
    - `coverage_map_generator.py`
    - `scenario_simulator.py`
    - Scenario datasets for all three cases

PHASE 2 (Weeks 7–9): INTEGRATION & VALIDATION

- Integrate with broadcast pipeline
- Verify RTCM flow and RTK end-to-end
- For each of the three test scenarios, compute:
    - Baseline/traditional metrics
    - AI-mode metrics under different intents

PHASE 3 (Weeks 10–12): PRODUCTION & OPTIMIZATION

- Code optimization
- Documentation
- Demo notebooks per scenario

4.3 TEAM 2: Broadcast Systems Engineer - RISHI

Responsibilities

PHASE 0 (Weeks 1–2): FOUNDATION & SETUP

- Study ATSC 3.0 specification (A/322, A/331, A/300)
- Install CommPy, reedsolo
- Verify libraries with unit tests

PHASE 1 (Weeks 3–7): ENCODERS, FEC, OFDM, PLPs

- ALP encoder
- FEC (LDPC + Reed-Solomon)
- OFDM modulator
- RF channel simulator
- PLP system (multiple PLPs)
- **Implement two configuration layers**:
    1. Traditional Profile Set (static)
    2. AI-Orchestrated Profile Application (dynamic)

PHASE 2 (Weeks 7–9): SCHEDULER, AI INTERFACE, METRICS

- Broadcast scheduler
- Controller that applies AI broadcast commands
- Metrics collector

PHASE 3 (Weeks 10–12): PRODUCTION & OPTIMIZATION

- Real-time targets (<50 ms per frame)
- Documentation & API reference

4.4 TEAM 3: AI/ML Systems Engineer - TARUNIKA

Responsibilities

PHASE 0–1 (Weeks 1–2): DATA PREP

- Data preprocessing pipeline for GNSS telemetry and broadcast metrics
- EDA, train/val/test splits
- **Define traditional baseline policy** (rule-based)

PHASE 1 (Weeks 3–6): NEURAL NETWORK DESIGN & TRAINING

- Architecture design (50+ inputs, 5+ outputs)
- Inputs include encoded **intent**

- Outputs target: redundancy factor, update frequency, tile resolution, FEC overhead, PLP allocation
- Training on 10K+ samples

PHASE 2 (Weeks 7–9): INFERENCE, EXPORT, AND COMPARISON LOGIC

- Inference engine
- ONNX export
- Scenario-specific evaluation for all three scenarios
- Build comparison notebooks

PHASE 3 (Weeks 10–12): PRODUCTION, API, & DEMO

- ONNX export, Qualcomm AI Hub integration
- API design
- Demo visualizations

4.5 12-Week Project Timeline

PHASE 0 (Weeks 1–2): FOUNDATION & SETUP

- Tool setup, basic RTK, basic ATSC and AI environments
- Define three canonical PoC test scenarios

PHASE 1 (Weeks 3–6): CORE MODULE DEVELOPMENT + SCENARIO DATA

- GNSS: RTCM + scenario data
- Broadcast: ALP + FEC + OFDM + PLPs
- AI: Data pipelines, baseline rule policies, model architecture

Milestone End of Week 6:

- All three scenarios implemented in simulation with:
  - Traditional broadcast + positioning configuration defined
  - Telemetry and metrics collection working

PHASE 2 (Weeks 7–9): INTEGRATION, AI TRAINING, AND COMPARISON

- E2E integration
- Model training & validation
- Scenario runs: Traditional vs AI for all scenarios
- Side-by-side results

PHASE 3 (Weeks 10–12): PRODUCTIONIZATION & DEMO

- Code hardening, performance optimizations, documentation
- Load testing with many simulated UEs
- Qualcomm AI Hub export

- **Demo Assembly:**
  - Scripts to replay each scenario (traditional and AI)
  - Plots and tables for each scenario
  - Narrative + slide content

4.6 Success Metrics & Acceptance Criteria

Scenario-Level Success

1. **Scenario 1 – High-Accuracy Drone in Rural Area**

   - Demonstrate that, under the **"Provide < 3 cm accuracy"** intent:
     - AI configuration meets or significantly improves accuracy vs traditional baseline
     - Spectrum usage remains within acceptable bounds
2. **Scenario 2 – Low-Bandwidth Rural ATSC Coverage**

   - Under **"Minimize ATSC spectrum usage"**:
     - AI reduces bits/s/Hz vs traditional
     - Accuracy and availability remain within acceptable bounds
3. **Scenario 3 – Urban Canyon Environment**

   - Under **"Guarantee service continuity in weak-signal regions"**:
     - AI improves FIX availability or reduces outage duration vs traditional
     - Any spectrum overhead is justified and visible

Demo Readiness

- For each scenario, you can:
  - Start in **traditional mode**, run, and show metrics
  - Switch to **AI-mode (same scenario, changed intent)**, run, and show improved KPI

PART 5: REFERENCE ARCHITECTURE & STANDARDS

5.1 ATSC 3.0 Reference Documents

Document	Purpose
A/331	To deliver positioning data (bitmap, RTK corrections) over ROUTE/DASH datacast
A/300	For service creation & delivery model
A/322	If you want to vary PLP robustness for "maximize accuracy" intent

5.2 Positioning-Specific Requirements

- **RTK correction format:** RTCM, MSM messages
- **Bitmap/feature-map data format:** Custom grid encoding
- **Timing info:** ATSC L1 signaling timestamps
- **GNSS → Broadcast synchronization mapping:** Frame alignment

5.3 AI-Intent Translation Examples

Intent	Broadcast Configuration
maximize accuracy	Higher bitrate, more frequent RTK updates
maximize reliability	Robust PLP, lower code rate
optimize bandwidth	Compress positioning deltas, reduce frequency

5.4 Required APIs

- Generating ROUTE/DASH for non-video data
- Setting PLP parameters (if using a modifiable transmitter)
- AI agent config control for the pipeline

PART 6: CODEBASE MODULES

6.1 Broadcast-Side Modules

1. **ALP encapsulator** (use libatsc3 or write Python/C++ wrapper)
2. **ROUTE/DASH packager**
3. **Service (SLT/LLS) descriptor generator**
4. **Transmitter control API** (if dynamic PLP/power needed)

5. **AI agent (Python preferred)**
    - Intent parser
    - Decision engine
    - Optimizer
    - Controller for ATSC parameters

6.2 Receiver-Side Modules

1. **ALP + ROUTE + DASH client** (libatsc3 or lightweight Python client)
2. **Positioning client** (RTK engine, bitmap decoder, fusion)

6.3 Orchestration Layer

1. REST/gRPC API exposing intents
2. Real-time analytics module
3. Telemetry aggregation

PART 7: KEY PERFORMANCE INDICATORS (KPIs)

KPI	Metric	Target
Positioning Accuracy	Horizontal/vertical error	3-10 cm
Reliability	Valid correction availability	>95%
Correction Delivery Latency	Broadcast → UE decode	<10 seconds
ATSC 3.0 Spectrum Efficiency	Bits/s/Hz for correction PLP	Maximize
Service Continuity	Dropout rate (rural/urban canyon)	<5%
Convergence Time	Time to FIX from signal loss	<30 seconds (AI) vs >60s (traditional)
Intent Fulfillment	% of time intent requirements met	>90%

PART 8: PHASE 0 – PREP & DATA (DETAILED)

Objective

Get tools, sample data, and baseline running.

Tasks

1. **Install RTKLIB** and confirm you can run a basic RTK solution on sample GNSS logs
    - Deliverable: `.pos` output files and a short report of baseline errors (no broadcast corrections)
2. **Collect or synthesize RTK correction frames (RTCM)**
    - RTKLIB sample RTCM or synthetic generator
3. **Create small bitmap tiles** for a small area (e.g., 100×100 binary tiles representing "good correction coverage" or landmark masks)
4. **Tip:** Use small synthetic scenarios to start (single road, a few tiles)

Motivation

- A broadcast-based correction delivery reduces per-vehicle bandwidth/redundancy compared to unicast — making RTK for many vehicles more efficient
- The system can support "mass-market" deployment of high-precision positioning for many vehicles, overcoming limitations of conventional NTRIP/unicast correction delivery
- Broadcast corrections + appropriate encoding and error robustness can maintain sufficient correction quality across a range of receivers, even under mobility and varying reception conditions

---

PART 9: MOTIVATION & BENEFITS

Why This Matters

1. Broadcast-Based Efficiency

Traditional unicast (NTRIP) delivery:

- Each vehicle maintains its own connection to a correction server
- Bandwidth scales linearly with number of vehicles
- Infrastructure cost increases with fleet size

Broadcast-based delivery (our approach):

- Single transmission serves unlimited vehicles in coverage area
- Bandwidth independent of fleet size
- Optimal for mass-market autonomous vehicle deployment

2. AI-Native Advantages

Traditional static systems:

- Cannot adapt to changing conditions
- One-size-fits-all approach wastes resources
- No feedback loop for continuous improvement

AI-native positioning-as-a-service:

- Intent-driven: Aligns with business goals
- Adaptive: Responds to real-time conditions
- Efficient: Right-sizes resources to actual needs
- Continuous improvement: Learns from fleet telemetry

3. ATSC 3.0 Opportunity

- Free spectrum (already licensed for TV broadcast)
- Wide coverage (30-50 km range)
- Mobile reception (works at highway speeds)
- Robust against fading and multipath
- Can be optimized per PLP for different use cases

PART 10: IMPLEMENTATION NOTES

Tools & Libraries

GNSS Team

- **RTKLIB**: Open-source RTK positioning library
  - GitHub: https://github.com/tomojitakasu/RTKLIB (https://github.com/tomojitakasu/RTKLIB)
  - License: BSD 2-Clause
- **Python 3.9+**: For scripting and data generation
- **NumPy, SciPy**: For numerical processing
- **PIL/Pillow**: For bitmap generation

Broadcast Team

- **CommPy**: For LDPC encoding
- **reedsolo**: For Reed-Solomon FEC
- **NumPy**: For OFDM signal processing
- **Python 3.9+**: Primary implementation language

AI Team

- **PyTorch 2.0+**: Neural network framework
- **ONNX**: Model export format
- **scikit-learn**: For preprocessing and validation
- **Pandas**: For data manipulation
- **Matplotlib/Seaborn**: For visualization

Development Environment

- **OS**: Linux (Ubuntu 20.04+) or macOS
- **Python**: 3.9 or higher
- **Git**: Version control
- **Docker**: (Optional) For containerization
- **Jupyter**: For exploratory analysis and demos

Repository Structure

```
ai-positioning-poc/
├── gnss/
│   ├── rtcm_generator.py
│   ├── coverage_map_generator.py
│   ├── scenario_simulator.py
│   └── tests/
├── broadcast/
│   ├── alp_encoder.py
│   ├── fec_encoder.py
│   ├── ofdm_modulator.py
│   ├── plp_system.py
│   ├── broadcast_scheduler.py
│   ├── broadcast_controller.py
│   └── tests/
├── ai/
│   ├── data_preprocessor.py
│   ├── intent_parser.py
│   ├── broadcast_decision_model.py
│   ├── inference_engine.py
│   └── tests/
├── integration/
│   ├── end_to_end_tests.py
│   └── scenarios/
│       ├── scenario1_drone_rural.py
│       ├── scenario2_low_bandwidth.py
│       └── scenario3_urban_canyon.py
├── notebooks/
│   ├── 01_gnss_analysis.ipynb
│   ├── 02_broadcast_simulation.ipynb
│   ├── 03_ai_training.ipynb
│   ├── 04_scenario_comparison.ipynb
│   └── 05_demo_walkthrough.ipynb
├── docs/
│   ├── ARCHITECTURE.md
│   ├── API_REFERENCE.md
│   ├── SCENARIOS.md
│   └── DEPLOYMENT.md
├── requirements.txt
├── setup.py
└── README.md
```

PART 11: RISK MITIGATION

Technical Risks

Risk 1: RTKLIB Integration Complexity

- **Mitigation**: Start with simple scenarios, use well-documented APIs, allocate extra time in Phase 0
- **Contingency**: Use pre-generated sample data if real-time generation proves difficult

Risk 2: ATSC 3.0 Simulation Accuracy

- **Mitigation**: Validate against known ATSC 3.0 test vectors, use conservative assumptions
- **Contingency**: Simplify RF channel model if full fidelity proves too complex

Risk 3: AI Model Training Time

- **Mitigation**: Use GPU acceleration, start training early (Week 4), use transfer learning if applicable
- **Contingency**: Fall back to rule-based system with simplified neural network

Risk 4: Integration Issues Between Teams

- **Mitigation**: Weekly sync meetings, clear API contracts, integration tests every 2 weeks
- **Contingency**: Broadcast engineer acts as integration coordinator

Risk 5: Demo Scenarios Don't Show Clear Improvement

- **Mitigation**: Design scenarios with known pain points for traditional systems, validate early
- **Contingency**: Adjust AI model or scenario parameters to ensure measurable improvement

Schedule Risks

Risk 1: Team Member Unavailability

- **Mitigation**: Cross-training, documentation, code reviews
- **Contingency**: Adjust scope or extend timeline by 1-2 weeks

Risk 2: Scope Creep

- **Mitigation**: Strict adherence to three defined scenarios, weekly scope reviews
- **Contingency**: Defer non-critical features to post-PoC phase

---

PART 12: QUALCOMM AI HUB DEPLOYMENT

Deployment Requirements

1. **Model Format**: ONNX with quantization
2. **Target Hardware**: Qualcomm Snapdragon (NPU/DSP)
3. **Performance**: <50ms inference latency
4. **Memory**: <100MB model footprint

Deployment Steps

1. **Export PyTorch model to ONNX**

```
torch.onnx.export(model, dummy_input, "model.onnx")
```

2. **Quantize model** (FP32 → INT8)

```
from onnxruntime.quantization import quantize_dynamic
quantize_dynamic("model.onnx", "model_quant.onnx")
```

3. **Upload to Qualcomm AI Hub**

   - Use AI Hub SDK
   - Benchmark on target hardware
   - Validate accuracy retention

4. **Integration testing**

   - Test on Snapdragon development board
   - Measure latency and power consumption
   - Verify end-to-end functionality

PART 13: COMPARISON METRICS SUMMARY

Scenario 1: High-Accuracy Drone in Rural Area

Metric	Traditional	AI-Native	Improvement
Accuracy (cm)	±5-8	±2-3	2-3x better
Convergence Time (s)	45-60	20-30	2x faster
Spectrum Usage (Mbps)	5 (constant)	3-7 (adaptive)	20% avg savings
FIX Availability (%)	85	95	+10%

Scenario 2: Low-Bandwidth Rural ATSC Coverage

Metric	Traditional	AI-Native	Improvement
Spectrum Usage (Mbps)	5 (constant)	2-3 (adaptive)	40-50% savings
Accuracy (cm)	±5-8	±8-12	Acceptable trade-off
Coverage Area (km²)	1000	1500	+50% with same resources
Intent Fulfillment	N/A	95%	Intent-driven

Scenario 3: Urban Canyon Environment

Metric	Traditional	AI-Native	Improvement
FIX Availability (%)	60	85	+25%
Outage Duration (s)	30-60	10-20	3x reduction
Multipath Mitigation	Basic	AI-enhanced	Qualitative improvement
Service Continuity (%)	90	99	+9%

PART 14: FUTURE ENHANCEMENTS (POST-POC)

Phase 2 Enhancements

1. **Extended Scenarios**

    - Highway high-speed mobility
    - Mixed urban/rural transitions
    - Multi-vehicle coordination

2. **Advanced AI Capabilities**

    - Online learning from fleet data
    - Predictive intent adjustment
    - Multi-objective optimization

3. **Additional Intent Types**

    - "Support emergency vehicles" (ultra-low latency)
    - "Optimize for battery life" (power-aware)
    - "Enable precise indoor-outdoor transition"

4. **Integration with 5G**

    - Hybrid broadcast + unicast
    - Network slicing for positioning
    - Low-latency augmentation via 5G

5. **Edge Intelligence**

    - Distributed AI inference at edge nodes
    - Local optimization based on regional conditions
    - Reduced backhaul requirements

PART 15: CONCLUSION

This PoC demonstrates a **paradigm shift** in precision positioning services:

- **From static to adaptive**: AI-driven real-time configuration
- **From resource-wasteful to efficient**: Intent-driven optimization
- **From one-size-fits-all to tailored**: Scenario-specific adaptation
- **From blind operation to informed**: Continuous feedback loop

The three canonical scenarios prove that:

1. **Scenario 1** shows AI can deliver superior accuracy when needed
2. **Scenario 2** shows AI can optimize spectrum usage intelligently
3. **Scenario 3** shows AI can maintain reliability in challenging environments

By integrating ATSC 3.0 broadcast, RTK-precision corrections, and AI-native orchestration, this system enables **mass-market deployment of centimeter-level positioning** for autonomous vehicles, drones, and robots.

Total Cost: $0 (100% open-source tools)

Timeline: 12 weeks

Team: 3 engineers

Outcome: Production-ready PoC with Qualcomm AI Hub deployment capability

---

APPENDIX A: GLOSSARY

- **ATSC 3.0**: Advanced Television Systems Committee standard for digital TV broadcast
- **FEC**: Forward Error Correction
- **GNSS**: Global Navigation Satellite System
- **IMU**: Inertial Measurement Unit
- **LDPC**: Low-Density Parity-Check (error correction code)
- **OFDM**: Orthogonal Frequency-Division Multiplexing
- **PLP**: Physical Layer Pipe (ATSC 3.0 concept for multiple services)
- **PPaaS**: Precise Positioning as a Service
- **RTK**: Real-Time Kinematic (high-precision GNSS)
- **RTCM**: Radio Technical Commission for Maritime Services (correction data format)
- **SSR**: State-Space Representation (GNSS correction method)
- **UE**: User Equipment (receiver device)

APPENDIX B: REFERENCE LINKS

- **RTKLIB**: https://github.com/tomojitakasu/RTKLIB (https://github.com/tomojitakasu/RTKLIB)
- **ATSC 3.0 Standards**: https://www.atsc.org/standards/ (https://www.atsc.org/standards/)
- **ITU-T FGAINN-I-097-R1**: Intent-driven orchestration framework
- **PyTorch**: https://pytorch.org/ (https://pytorch.org/)
- **ONNX**: https://onnx.ai/ (https://onnx.ai/)
- **Qualcomm AI Hub**: https://aihub.qualcomm.com/ (https://aihub.qualcomm.com/)

APPENDIX C: CONTACT & SUPPORT

For technical questions or collaboration:

- **Project Repository**: [To be created]
- **Team Lead**: [To be assigned]
- **Technical Support**: [To be defined]

Document End

Version: 3.1 (Final)

Date: January 19, 2026

Status: Ready for Implementation