Jetson Nano ML Training System

# Complete System Documentation

Generated: September 17, 2025

Version: 1.0

System: Multi-Player Athlete Monitoring & ML Training

# Table of Contents

1. 1. System Overview
2. 2. Complete Data Flow Analysis
3. 3. Training Data Generation Flow
4. 4. Live Prediction Flow
5. 5. Key Integration Points
6. 6. Conflict Prevention System
7. 7. File Structure
8. 8. Configuration Management
9. 9. Installation & Setup
10. 10. Usage Examples
11. 11. Troubleshooting

# 1. System Overview

The Jetson Nano ML Training System is a comprehensive athlete monitoring solution that provides real-time performance analysis and automated machine learning model training. The system is designed to handle multiple athletes simultaneously while maintaining data integrity and preventing operational conflicts.

## Key Features

* • Multi-player sensor data simulation (1-30 athletes)
* • Real-time prediction engine for live performance monitoring
* • Automated ML model training with conflict prevention
* • Health metrics calculation (heart rate, stress, VO2 Max, TRIMP)
* • Session management with automatic data saving
* • Conflict prevention between training and prediction
* • Scalable architecture supporting up to 30 athletes
* • GPU acceleration support for Jetson Nano

## Key Components

|  |  |  |
| --- | --- | --- |
| Component | Purpose | File |
| Data Publisher | Generates realistic sensor data | publisher.py |
| Prediction Engine | Real-time ML predictions | test\_30\_players.py |
| Training System | Automated ML model training | sup\_ml\_rf\_training.py |
| Configuration | Unified system settings | jetson\_config.yaml |
| Deployment Helper | Setup and management | jetson\_deploy.py |

# 2. Complete Data Flow Analysis

## Flow 1: Training Data Generation & ML Training

1. PUBLISHER (Training Data Generation)  
 • Command: python publisher.py <num\_players>  
 • Generates: Realistic sensor data (10 Hz sampling rate)  
 • MQTT Topics: player/{device\_id}/sensor/data  
 • Data includes: device\_id, timestamp, athlete\_id, age, weight, height, gender,   
 accelerometer (acc\_x/y/z), gyroscope (gyro\_x/y/z), magnetometer (mag\_x/y/z), heart\_rate\_bpm  
 • Storage: In-memory collection during session  
 • On Stop (Ctrl+C): Automatically saves to CSV files  
   
 2. DATA SAVING (Automatic on Publisher Stop)  
 • Directory: athlete\_training\_data/player\_{id}/  
 • Filename Pattern: TR{seq}\_A{athlete\_id}\_D{device\_id}\_{timestamp}.csv  
 • CSV Format: timestamp,athlete\_id,age,weight,height,gender,acc\_x,acc\_y,acc\_z,gyro\_x,gyro\_y,gyro\_z,heart\_rate  
 • Sequence Management: Auto-increments (TR1, TR2, TR3...)  
 • Status: Ready for ML training  
   
 3. ML TRAINING (Triggered by new data)  
 • Command: python sup\_ml\_rf\_training.py  
 • Conflict Check: Ensures no live prediction running  
 • Player Scan: All 30 players for new/updated TR files  
 • Decision Logic: Train if no model exists OR new data OR force\_retrain=true  
 • Data Processing: Latest 3 TR sessions per player with feature engineering  
 • Model Training: RandomForest (100 trees, max\_depth=8, min\_samples\_split=5)  
 • Model Saving: PKL format and Hummingbird format for GPU acceleration  
 • Result: Updated models ready for prediction

## Flow 2: Live Prediction During Actual Play

1. PUBLISHER (Real Sensor Data)  
 • Source: Real athlete sensors OR simulated data  
 • Command: python publisher.py <num\_players>  
 • Data Rate: 10 Hz per player  
 • Multi-player: Supports 1-30 athletes simultaneously  
   
 2. PREDICTION ENGINE (test\_30\_players.py)  
 • Startup: Creates lockfile, connects to MQTT  
 • Model Loading: Discovers and loads player-specific models  
 • Real-time Processing: Sensor fusion, motion analysis, ML prediction  
 • Health Metrics: Stress, VO2 Max, TRIMP, Energy Expenditure, G-impact Detection  
 • Session Management: Multi-device support with separate processing contexts  
 • Outputs: Real-time JSON metrics, session logs, G-impact event logs

# 3. Conflict Prevention System

The system implements a robust conflict prevention mechanism to ensure training and prediction  
 operations do not interfere with each other. This prevents model corruption, prediction   
 instability, and resource conflicts.

## Implementation Details

• Process Detection: Scans running processes for prediction scripts  
 • Lockfile Mechanism: Creates .prediction\_running.lock during prediction  
 • User Interaction: Prompts user when conflicts detected  
 • Timeout Protection: 5-minute maximum wait time  
 • Automatic Cleanup: Removes lockfiles on exit or interruption  
 • Configuration Control: Can be disabled or customized via config file

# 4. File Structure

The system maintains a well-organized directory structure that separates different  
 types of data and maintains clear relationships between components.

## Core Directories

• Core Scripts: publisher.py, test\_30\_players.py, sup\_ml\_rf\_training.py

• Configuration: jetson\_config.yaml (unified configuration)

• Training Data: athlete\_training\_data/player\_X/TR\*.csv

• Models: athlete\_models\_pkl/ (scikit-learn), athlete\_models\_tensors\_updated/ (Hummingbird)

• Runtime Data: logs/ (session logs), A\*\_\*/ (real-time outputs)

• System Files: lockfiles, training logs

# 5. Configuration Management

All system settings are centralized in jetson\_config.yaml, providing unified control  
 over training parameters, feature engineering, conflict prevention, logging, and  
 Jetson-specific optimizations.

## Key Configuration Sections

• paths: Data and model locations

• training: ML algorithm parameters

• feature\_engineering: Advanced feature settings

• jetson: GPU/CPU optimization settings

• prediction\_check: Conflict prevention configuration

• logging: Log levels and file locations

• monitoring: System monitoring parameters

# 6. Installation & Setup

## System Requirements

• Hardware: NVIDIA Jetson Nano (or compatible system)

• OS: Ubuntu 18.04+ or Jetson Linux

• Python: 3.7+

• Memory: Minimum 4GB RAM

• Storage: 16GB+ available space

## Installation Steps

1. Navigate to project directory: cd /path/to/jetson\_nano\_test  
 2. Install dependencies: pip install -r requirements.txt  
 3. Verify installation: python jetson\_deploy.py check  
 4. Setup directories: python jetson\_deploy.py setup  
 5. Validate configuration: python jetson\_deploy.py validate  
 6. Start MQTT broker: sudo systemctl start mosquitto

# 7. Usage Examples

## Training Data Generation

• All 30 players: python publisher.py 30  
 • 5 random players: python publisher.py 5   
 • 1 random player: python publisher.py 1  
   
 Let the publisher run for desired duration, then press Ctrl+C to save data.

## ML Model Training

• Automatic training: python sup\_ml\_rf\_training.py  
 • Check system status: python jetson\_deploy.py status  
 • Run with helper: python jetson\_deploy.py run

## Live Prediction

Multi-player prediction:  
 Terminal 1: python test\_30\_players.py  
 Terminal 2: python publisher.py 10  
   
 Single-player prediction:  
 Terminal 1: python test\_deployment1.py  
 Terminal 2: python publisher.py 1

# 8. Troubleshooting

## Common Issues

• MQTT Connection Failed: Check mosquitto service: sudo systemctl status mosquitto

• Training Won't Start: Check for running prediction: ps aux | grep test\_30\_players

• No Models Found: Run training first: python sup\_ml\_rf\_training.py

• Memory Issues: Reduce batch\_size in jetson\_config.yaml

• Permission Errors: Check file permissions: chmod 755 .

• Model Loading Errors: Retrain models: rm athlete\_models\_tensors\_updated/\*

# 9. System Specifications

## Performance Specifications

• Sampling Rate: 10 Hz per player

• Training: Handles 30 players with 3 sessions each

• Prediction: Real-time processing for up to 30 players

• Latency: <100ms prediction time per player

• Memory: <2GB RAM usage (configurable)

• Storage: ~1MB per training session per player

This documentation covers the complete Jetson Nano ML Training System for athlete   
 monitoring and performance prediction. The system provides a comprehensive solution   
 for real-time athlete performance analysis with automated machine learning capabilities.  
   
 For additional support or feature requests, refer to the individual script documentation   
 and configuration files in the project directory.