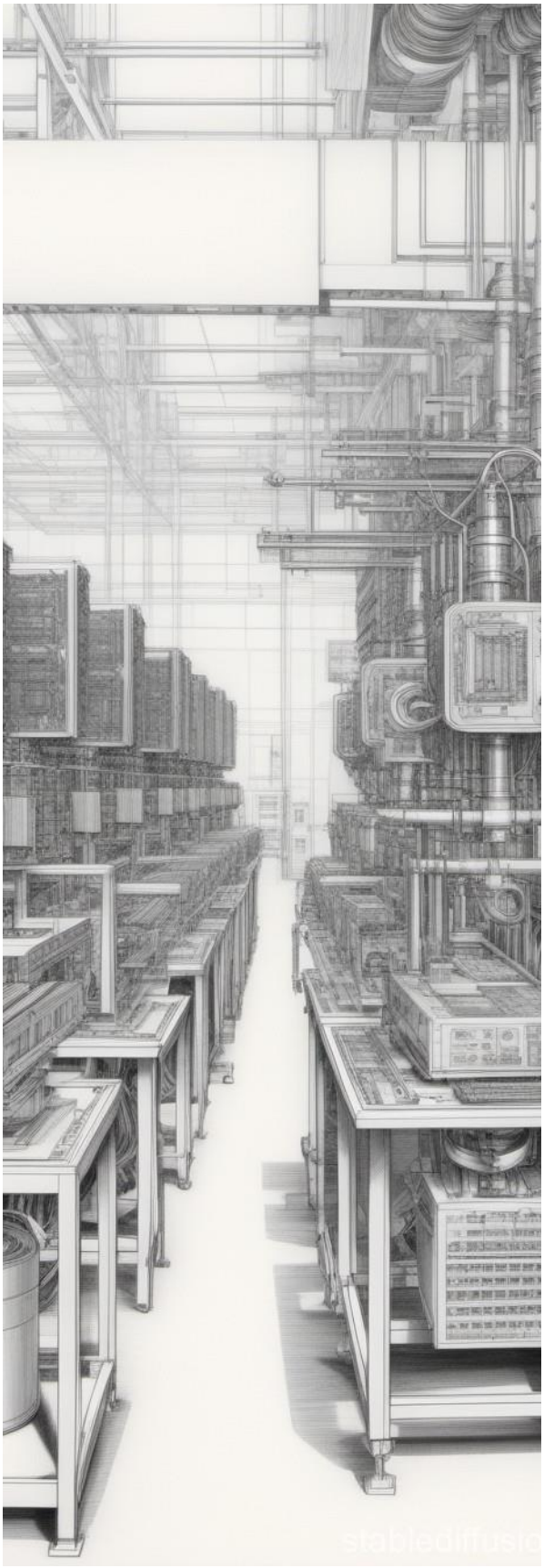
Prepared for: Dr. Humera Noor
humeranoor.minhas@ue-germany.de

Prepared by: Can Özkan
can.ozkan@ue-germany.de
can.ozkan.de@gmail.com
+49 15510201097
MSc Software Engineering
University of Europe for Applied Sciences
Student ID: 59301633

CAPSTONE PROJECT PROPOSAL



PROJECT TITLE

ROS2-Based Smart Factory Digital Twin: Predictive Maintenance and Dynamic Production Flow with Edge AI.

I. EXECUTIVE SUMMARY

The system simulates machines in a production line, generates synthetic sensor data from these machines, sends the data to a machine learning (ML)-based predictive maintenance model and feeds preventive actions back into the production process via ROS2. As a result, *on a virtual production line*, production planning, sensor data flow, model predictions, load sharing and preventive maintenance decisions are processed in real-time.

The scope is limited to 6 machines/stations (scalable, with an MVP starting with 2 machines). The system is fully virtual (with optional Gazebo for 3D simulation) and can later integrate Raspberry Pi for edge testing. Data uses synthetic generation plus open-source datasets (e.g., NASA CMAPSS or Fraunhofer Hydraulic Test Rig). Target metrics include RUL (Remaining Useful Life) prediction accuracy >80%, latency <50ms per node and a 25% reduction in simulated downtime.

Key components include a production line with sensor and controller nodes per machine, job order scheduling in JSON format, synthetic data generation based on drift and noise models, an edge AI predictor node using XGBoost or LSTM for RUL estimation, controller nodes for preventive actions like load reduction or job redistribution and a Streamlit dashboard for real-time monitoring via `rosbridge_suite`. The system operates in a closed-loop, adaptively adjusting behavior based on ML insights to create a responsive digital twin. Deliverables include a working ROS2 simulation, trained ML model, dashboard, edge performance measurements, GitHub repository, video demo and technical report.

II. BACKGROUND

In modern manufacturing, smart factories leverage digital twins to mirror physical processes virtually, enabling real-time optimization and predictive capabilities. This project addresses inefficiencies in traditional production lines, such as unplanned downtime due to machine failures, by integrating ROS2 for orchestration, synthetic sensor simulation and edge AI for proactive maintenance.

The core problem is the lack of adaptive, real-time systems that predict machine degradation and dynamically reroute production flows. The proposed solution simulates a production line where machines generate sensor data (e.g., pressure, temperature, vibration), which feeds into ML models for RUL predictions. Preventive actions, such as reducing load or pausing jobs, are then applied to minimize disruptions.

Project goals include:

- **Developing a scalable ROS2-based simulation limited to 6 machines (MVP with 2).**
- **Achieving >80% RUL accuracy and <50ms latency.**
- **Demonstrating a 25% downtime reduction in simulations.**
- **Enabling optional edge deployment on Raspberry Pi for low-latency inference.**

This aligns with software engineering principles of modularity, real-time systems and AI integration, contributing to Industry 4.0 advancements.

III. METHODOLOGY & TECHNICAL APPROACH

The project follows an Agile development lifecycle with iterative sprints: requirements gathering via JSON job definitions, prototyping of ROS2 nodes, ML model training offline and evaluation through simulated runs.

III.I High-Level Architecture

The system comprises modular ROS2 nodes communicating via topics (e.g., `/job_orders`, `/machine_X/sensors`, `/machine_X/predictions`). Technologies include ROS2 (Jazzy) for messaging, Python (rclpy) for nodes, scikit-learn/PyTorch/TensorFlow Lite for ML, Streamlit for dashboard and optional Gazebo for 3D visualization, MQTT for external alerts and JSON/YAML for configurations.

III.II Production Line

Limited to 6 machines/stations (e.g., Hydraulic Press, CNC Machine), each with sensor_node (data producer) and controller_node (action applicator).

Example sensors and ranges:

Machine/Station (Examples)	Sensors (Example Value Ranges)	Notes
Hydraulic Press	Pressure (0-100 bar), Temperature (20-80°C), Current (0-50A), Vibration (0-10g)	High pressure focus, fault: Excessive vibration.
CNC Machine	RPM (0-5000 rpm), Temperature (20-100°C), Load (0-100%), Vibration (0-5g)	RPM focus, fault: RPM fluctuations.
Industrial Fan	RPM (0-3000 rpm), Temperature (10-60°C), Current (0-20A)	Cooling focus, fault: Current increase.
Assembly Robot	Current (0-30A), Load (0-50kg), Vibration (0-8g)	Motion focus, fault: Load imbalance.
Quality Control Unit	Temperature (15-70°C), Load (0-20%), Vibration (0-3g)	Inspection focus, fault: Vibration spikes.
Packaging Station	RPM (0-2000 rpm), Load (0-10kg), Temperature (20-50°C)	Final stage, fault: RPM drop.

A central predictor_node runs on Raspberry Pi (simulated edge with CPU limits).

III.III Job Orders & Production Flow

Job Scheduler Node receives JSON orders (e.g., `{"job_id": "J123", "sequence": ["Press", "CNC", "QC", "Packaging"], "workload": 75, "estimated_time": 300}`) and publishes to `/job_orders`. Machines complete tasks via `/machine_X/completed`, triggering the next in sequence. On faults, loads redistribute.

III.IV Synthetic Data Generation

Sensors produce data every 0.5s using models like `temp = baseline + drift_rate * t + gaussian_noise` (NumPy), based on CMAPSS distributions. Baseline per machine (e.g., press pressure: 50 bar), drift for degradation, 5% Gaussian noise (mean=0, std=0.1). Published to `/machine_X/sensors`; fault injection at t=100 (e.g., vibration x2).

III.V Predictive Maintenance Model

XGBoost/LSTM for RUL regression (offline training on CMAPSS, 80/20 split, 5-fold CV, MAE <10%). Features from 30s windows (mean/std/variance). ONNX for edge optimization (<20ms inference). Subscribes to sensor topics, publishes to `/machine_X/predictions`.

Edge simulation: Thread limit=2, CPU 50% cap; log latency (time.perf_counter) and power (simulated: 5W base + 0.1W/inference).

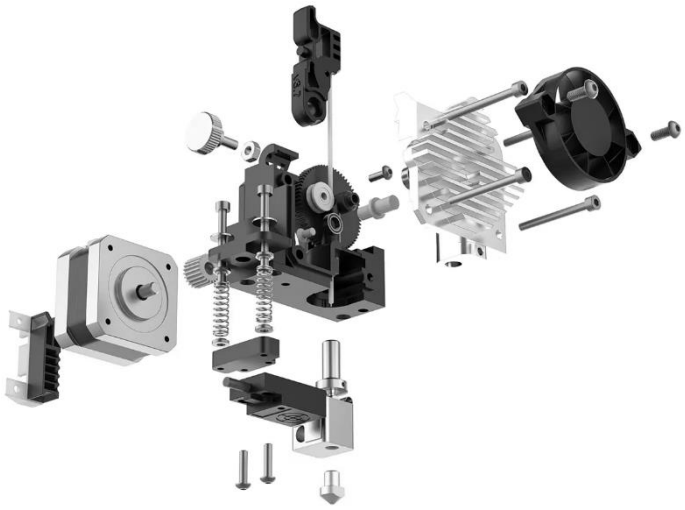
Optional RPi deploy with rclpy/TFLite, WiFi DDS; metrics: <50ms inference, <5% accuracy drop.

III.VI Preventive Actions

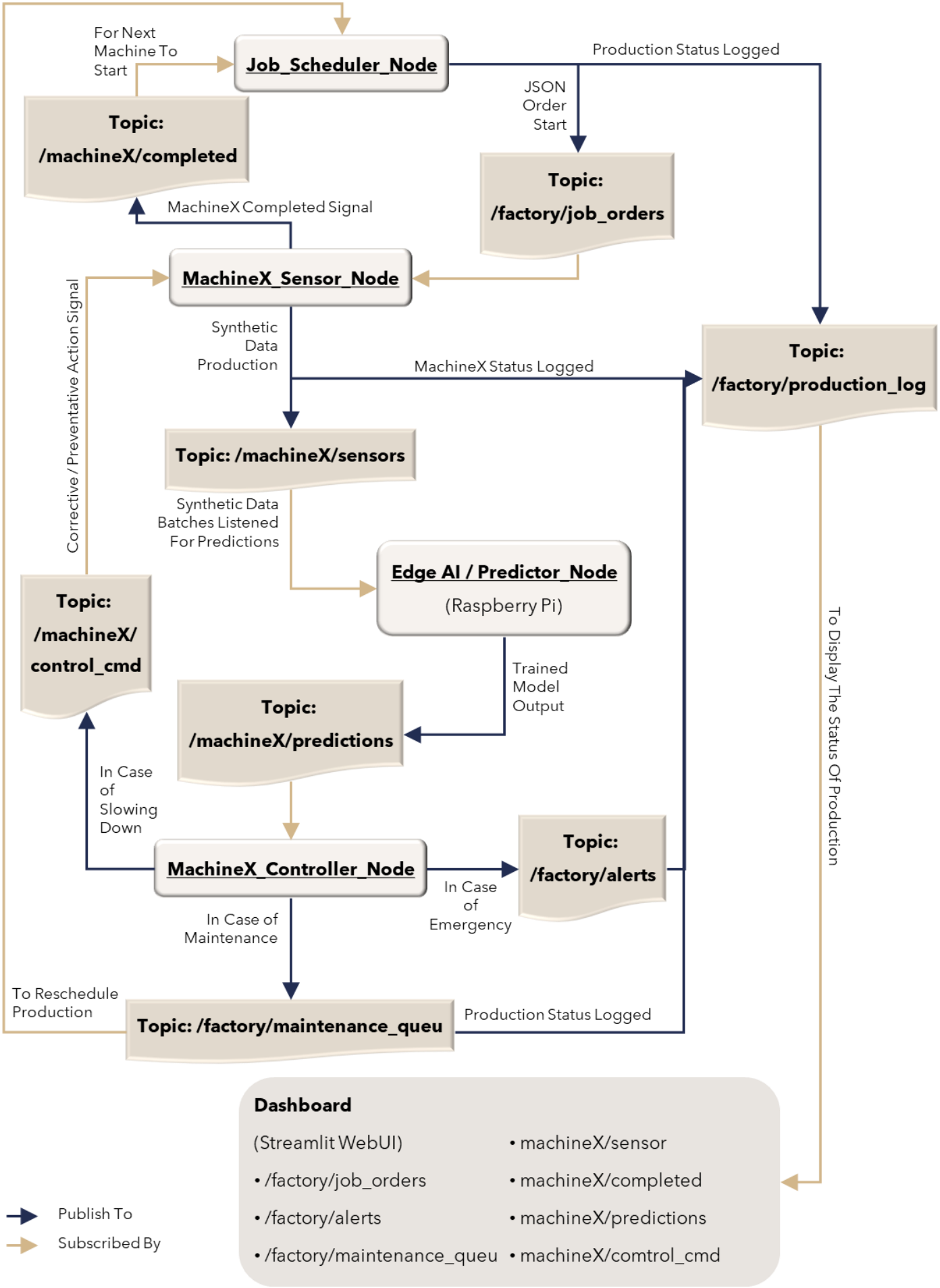
Controller Node subscribes to predictions, applies actions (e.g., REDUCE_LOAD lowers load sensor, PAUSE_JOB halts on high temperature, REDISTRIBUTE_JOB reroutes). Feedback via `/machine_X/control_cmd` modifies sensor dynamics (e.g., reduce drift_rate), creating a closed loop.

III.VII Dashboard & Monitoring

Streamlit interface connects via rosbridge_suite WebSocket, displaying real-time machine status, sensor graphs, active jobs, RUL values and actions. Subscribes to `/machine_X/sensors, /machine_X/predictions, /factory/alerts`.



III.VIII Data & Communication Flow



Functional requirements: Real-time data flow, ML inference, action feedback.

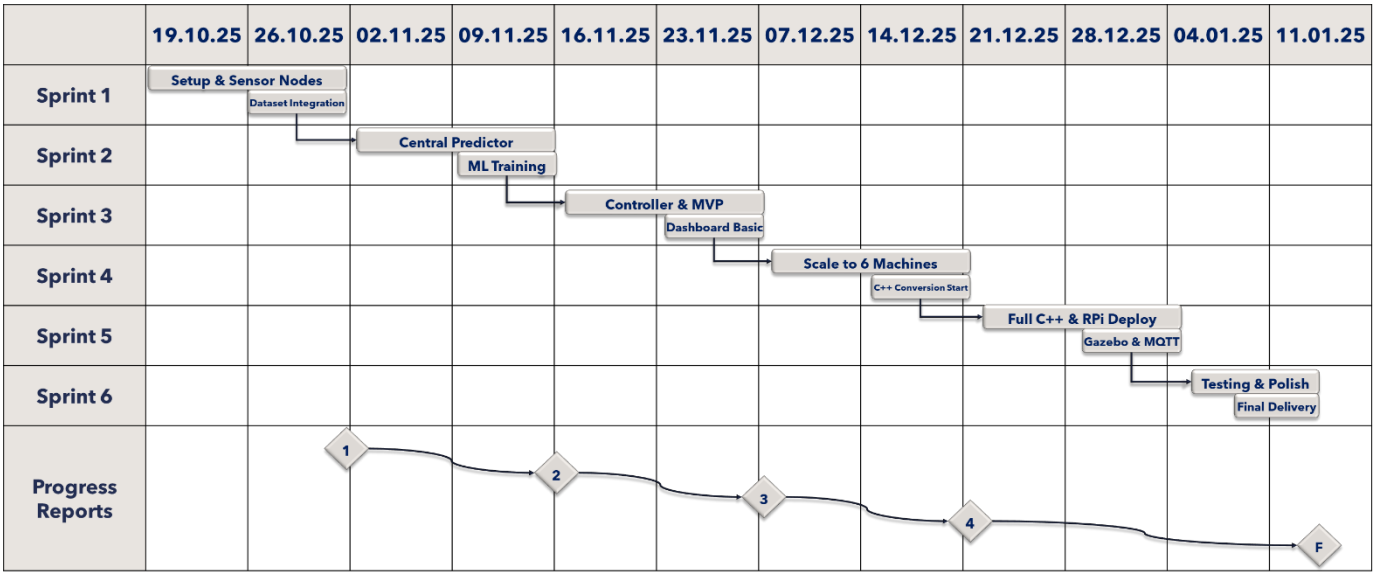
Non-functional: Latency <50ms, scalability to 6 machines, modularity for extension.

Testing: Unit/integration via ROS2 tools; evaluation on RUL accuracy, latency logs, simulated downtime.

IV. Timeline/Schedule

Sprint	Date Range	Goals	Deliverables	Risks & Mitigations	Progress Report
1 Setup & Data Generation	Oct 17-Oct 30, 2025	• Validate ROS2 Jazzy environment • Set up 2 machines (Hydraulic Press + CNC) sensor nodes in Python • Integrate synthetic data generator (CMAPSS/Fraunhofer drift + noise) • Initialize simple JSON-based job scheduler	• 2 sensor nodes (`rclpy`, `pub /machine1/sensors`) • Dataset merge script (Pandas, 80% accuracy benchmark) • Basic launch file (ros2 launch)	Dataset mismatch *Mitigation:* Pandas feature alignment	GitHub push (sensor repo), short demo video
2 Central Predictor & ML Integration	Oct 31 - Nov 13, 2025	• Add central predictor node (TFLite + XGBoost RUL model) • Subscribe to all sensor topics and run inference • Ack sync module	• Central node (`sub /machineX/sensors`, `pub /machineX/predictions`) • Offline ML training script (RUL MAE <10%) • Latency log (`perf_counter`, target <50 ms)	Inference latency *Mitigation:* Try quantization	Demo video (data > prediction flow)
3 MVP Completion & Closed-Loop	Nov 14 - Nov 27, 2025	• Add controller nodes for each machine (Python) • Implement actions (e.g., REDUCE_LOAD feedback) • Test job sequence (2-machine chain) • MVP benchmark: simulate 25% downtime reduction	• 2 controller nodes (`sub predictions`, `pub control_cmd`) • End-to-end MVP (job > loop > log) • Basic Streamlit dashboard (live sensor chart)	Sync blocking *Mitigation:* Ack timeout retry	MVP demo video, close Gantt milestone
4 Scaling & Start of C++ Conversion	Nov 28 – Dec 11, 2025	• Scale system to 6 machines • Convert sensor + controller nodes to C++ (rclcpp, basic pub/sub) • Add load balancing with PuLP	• 6-machine integration (topic wildcard) • C++ version: 1 sensor node (Press, 50% conversion) • Basic optimization (downtime metric)	C++ debugging *Mitigation:* Use ROS2 tutorials + gdb	C++ prototype push, hybrid Python/C++ demo

Sprint	Date Range	Goals	Deliverables	Risks & Mitigations	Progress Report
5 Advanced Features & Edge Testing	Dec 12 – Dec 26, 2025	• Convert remaining nodes to C++ (controller + partial predictor) • Deploy to RPi (WiFi DDS, real latency test) • Integrate Gazebo (optional 3D visualization) • Add MQTT export	• Full C++ conversion (~80% nodes) • RPi edge test report (power/latency log) • Extended dashboard (RUL table + alerts)	RPi connectivity *Mitigation:* Ethernet fallback	RPi demo video, final benchmarks
6 Polish, Testing & Delivery	Dec 27, 2025 – Jan 5, 2026	• Conduct system-wide testing (10 healthy/faulty scenarios) • Prepare documentation (presentation, diagrams) • Polish GitHub (README, CI/CD basics) • Final presentation video (15 min) + 3 min demo video	• Unit tests (`pytest` + `gtest`) • Technical report (PDF, including challenges) • Final delivery package (repo + video)	Time pressure *Mitigation:* Revert to MVP scope	Final presentation + LinkedIn showcase



V. Resources, Constraints, Risks & Mitigation

V.I Resources

- **Software**: ROS2 (Jazzy), Python rclpy, scikit-learn/PyTorch/TensorFlow Lite, Streamlit, Gazebo (optional), MQTT (optional), NumPy for data gen, ONNX for optimization, psutil for logging.
- **Hardware**: Fully virtual (no physical required); optional Raspberry Pi for edge testing.
- **Data**: Synthetic + NASA CMAPSS/Fraunhofer datasets.
- **Budget**: Minimal (open-source tools); solo development.

V.II Constraints

Limited to 6 machines (MVP: 2); fully virtual with optional real sensor injection; synthetic data for consistency.

V.III Risks & Mitigation

Technical: ML accuracy <80%, Mitigate with 5-fold CV and fallback anomaly detection.

Timeline: Integration delays, Use Agile sprints with weekly checkpoints and biweekly reports.

Edge latency >50ms: Simulate CPU limits early; optimize with 8-bit quantization.

Data realism: Calibrate noise/drift against CMAPSS; optional USB sensor for validation.

VI. Expected Results and Dissemination

Anticipated outcomes:

- Working ROS2-based production simulation.
- Trained predictive maintenance model (RUL >80% accuracy).
- Streamlit dashboard for live monitoring.
- Edge device performance measurements (latency <50ms, simulated power).

Success criteria: 25% downtime reduction in simulations, modular code for extension.

Dissemination: GitHub repository with source code, video demo of full flow, technical report (PDF).

VII. Conclusion

This project demonstrates a feasible, innovative ROS2-based digital twin for smart factories, integrating edge AI for predictive maintenance and dynamic flows. By achieving real-time adaptability in a fully virtual setup, it contributes to scalable software engineering solutions for Industry 4.0, ready for approval and implementation.