Tractor Health Monitoring System – Project Report

(Buhler Versatile Inc.)

Table of Contents

Executive Summary

This report details the design, development, and deployment of a robust, real-time Tractor Health Monitoring System for D-series 4WD tractors. Addressing the critical need for proactive maintenance and minimized unscheduled downtime in demanding agricultural environments, this solution seamlessly integrates Machine Learning (ML), Internet of Things (IoT), sophisticated Embedded Systems, and scalable Cloud Computing. The primary objective is to enable early detection and classification of mechanical faults and abnormal operational behavior directly at the edge, supported by an automated, continuous cloud-based ML lifecycle.

The system's architecture is engineered across multiple layers. At the edge, a custom-fabricated Central Processing Unit (CPU) combines an Arduino microcontroller for high-frequency sensor data acquisition with a Raspberry Pi single-board computer for local data processing and ML inference. This CPU is housed within a custom-designed, IP67-rated enclosure, ensuring resilience against the extreme dust, moisture, vibration, and temperature fluctuations inherent to farm tractor operations. Connectivity to industrial-grade sensors positioned across critical tractor subsystems (engine, transmission, hydraulics, alternator etc.) is achieved via a rugged, custom-built wiring harness utilizing Deutsch connectors, guaranteeing robust and sealed electrical integrity. Data communication involves both serial interfaces and the tractor's CAN Bus (J1939 protocol), managed by an MCP2515 module for parameter acquisition. On-device software performs real-time data preprocessing and executes lightweight TensorFlow Lite (TFLite) models: a LSTM autoencoder for unsupervised anomaly detection, followed by a supervised CNN+LSTM classifier for specific fault categorization, providing immediate, actionable insights.

In the cloud, a fully automated MLOps pipeline, orchestrated on Google Cloud's Vertex AI using the Kubeflow Pipelines (KFP) SDK, ensures continuous model improvement and operationalization. Daily raw sensor data, uploaded from the edge to Google Cloud Storage (GCS) with timestamped versioning, is then integrated into BigQuery, forming a scalable historical dataset. Each step of this retraining pipeline—from data preparation and feature engineering to model training, evaluation, and TFLite export—is containerized using Docker images, for reproducibility and consistent execution environments. The newly trained and optimized models are automatically pushed back to GCS. A dedicated update mechanism on the edge device (`update_models.py` script), triggered by a cron job, periodically pulls these latest models, performing a replacement and service restart to ensure the on-tractor intelligence is always current. Furthermore, critical anomaly alerts

and telemetry data are streamed from the edge to the cloud via Google Cloud Pub/Sub, enabling centralized monitoring and potential downstream integrations.

A modular, touch-friendly Graphical User Interface (GUI), developed using PyQt5, provides in-cabin operators with a clear, real-time visualization of tractor health, highlighting anomalies and classified faults through intuitive color-coding and dynamic updates.

This project represents a sophisticated integration of ML Engineering (orchestration, operationalization, productionalization), Embedded Systems, IoT, Cloud ML, Computer Engineering, Data Engineering, and Real-time Computing. It delivers a production-ready, intelligent, and fault-aware system poised to transform tractor maintenance from reactive to predictive, significantly enhancing operational efficiency and asset longevity in the agricultural sector.

2. Introduction

2.1. Project Motivation

Modern agricultural machinery, particularly D-series 4WD tractors, operates under exceptionally demanding environmental and operational conditions. Failures in critical tractor subsystems, such as the engine, hydraulic system, brakes, or electrical components, can result in severe consequences, including damage to the million-dollar equipment, costly unscheduled downtime, significant yield losses, and potential safety hazards. Traditional maintenance approaches, which often rely on reactive repairs after a breakdown or rigid, time-based preventative schedules, are inherently inefficient and insufficient for detecting early-stage mechanical issues.

The rapid advancements in low-cost embedded systems, industrial-grade sensors, and scalable cloud computing platforms now present a unique opportunity to develop intelligent, proactive health monitoring solutions for agricultural equipment. However, the integration of such a comprehensive system—encompassing raw data acquisition from ruggedized hardware, complex CAN Bus communication, sophisticated cloud-based machine learning (ML) pipelines, and real-time user feedback—poses significant engineering challenges across embedded system design, data engineering, and machine learning operationalization.

This project is specifically motivated by the need to engineer a complete, end-to-end automated system capable of continuously monitoring the health of critical tractor subsystems in real-time. The system aims to provide accurate fault detection, precise anomaly identification, and actionable feedback directly to operators. The overarching vision is to transition from reactive or time-based maintenance to a data-driven, condition-based predictive maintenance paradigm, ultimately reducing unplanned downtimes, optimizing maintenance schedules, and ensuring more efficient fleet and operations management within the agriculture industry.

Figure 2-1 An image of a 4WD tractor

2.2. Objectives

The primary objectives guiding the development of the Tractor Health Monitoring System are as follows:

1. **Real-time Data Acquisition:** To design and construct a resilient edge computing platform capable of acquiring high-fidelity multi-sensor data from a D-series 4WD agricultural tractor, including both industrial sensor readings and internal CAN Bus (J1939) data.
2. **Integrated Embedded Systems:** To implement a sophisticated embedded system for industrial sensor integration, encompassing temperature sensors, float level switches, and 3-axis accelerometers, with low-level data handling orchestrated by an Arduino microcontroller and higher-level processing on a Raspberry Pi.
3. **Advanced Communication Protocol Implementation:** To enable communication over the J1939 CAN Bus protocol, specifically integrating an MCP2515-based CAN receiver module for reliable digital data acquisition from the tractor's internal electronic control units (ECUs).
4. **Automated End-to-End MLOps Pipeline:** To develop and deploy a fully automated, scalable end-to-end ML pipeline that performs data ingestion, data preprocessing/feature engineering, model training, model evaluation and model serving on Google Cloud Platform (GCP) using Vertex AI Pipelines and the Kubeflow Pipelines (KFP) SDK. This pipeline will facilitate continuous model retraining,

validation, and evaluation, supporting both unsupervised anomaly detection via autoencoders and supervised fault classification.

5. **Efficient Edge ML Inference:** To perform real-time ML inference directly on the edge device (Raspberry Pi) using TFLite-optimized models, ensuring low-latency detection of anomalies and classification of faults as they occur.
6. **Intuitive Operator Interface:** To create a user-friendly Graphical User Interface (GUI) for in-cabin operator interaction, capable of displaying real-time sensor data, clear health status indicators, and actionable fault messages.
7. **Ruggedized Hardware Deployment:** To design and fabricate a highly durable, custom-built IP67-rated wiring harness utilizing industrial-grade Deutsch connectors, and to assemble an IP67-rated enclosure box for the central processing unit, ensuring robust and reliable operation in harsh agricultural conditions.
8. **Comprehensive System Validation:** To validate the entire system through extensive simulation and bench testing, followed by potential thorough field testing on a D-series 4WD tractor to confirm real-world performance and reliability.

2.3. System Overview

The Tractor Health Monitoring System employs a layered and modular architecture, integrating specialized hardware, robust software, advanced machine learning, and scalable cloud infrastructure components to achieve its objectives.

Embedded Layer (Edge Device): The foundational layer resides on the tractor. Industrial-grade sensors (e.g., temperature, accelerometers, float level switches) capture raw operational signals from various critical tractor components.

An Arduino Mega microcontroller is responsible for low-level, high-frequency single raw data sample acquisition and communication of the data to a Raspberry Pi over a USB serial connection. Concurrently, a CAN Bus transceiver (MCP2515 module) is integrated with the Raspberry Pi to receive critical digital data streams from the tractor's internal J1939 CAN Bus network.

Edge Processing and Inference: The Raspberry Pi serves as the primary edge processing unit. It executes Python-based data acquisition scripts that aggregate and synchronize the incoming sensor and CAN data. This combined dataset is uploaded to Google Cloud Storage (GCS) to be used for continuous model improvement, and the single raw data concurrently undergoes real-time preprocessing, including scaling and feature engineering, before being fed into a lightweight ML inference engine. This engine utilizes

TFLite-optimized models: an autoencoder performs unsupervised anomaly detection on the incoming data stream, and if an anomaly is identified, a supervised classifier model attempts to pinpoint the specific fault condition. Real-time sensor readings and inference results are displayed on a dynamic in-cabin GUI, built using PyQt5, providing operators with immediate visual feedback on the tractor's health status.

Cloud Training and MLOps Pipeline: The cloud infrastructure, built on Google Cloud Platform, provides the scalable backbone for continuous model improvement. Raw sensor and CAN data, collected locally on the Raspberry Pi, is periodically uploaded to Google Cloud Storage (GCS) with timestamped versioning (e.g., daily batches). This data is then ingested and consolidated into Google BigQuery, serving as the comprehensive historical dataset for ML model training. A fully automated, end-to-end ML pipeline, defined using the Kubeflow Pipelines (KFP) SDK and orchestrated on Vertex AI Pipelines, handles the entire retraining workflow. This pipeline includes components for data ingestion (reading from BigQuery), data preprocessing, model training (both autoencoder and classifier), evaluation, and TFLite model export. Crucially, each pipeline component is containerized using Docker, ensuring environmental consistency and reproducibility across development and production stages. Newly trained and validated TFLite models are then deployed back to GCS, ready for serving.

Model Deployment and Communication: A robust cloud-to-edge model update mechanism ensures the edge device always operates with the latest intelligence. A dedicated script on the Raspberry Pi (`update_models.py`), scheduled via a cron job, periodically checks GCS for newer model versions. Upon detection, it securely downloads and atomically replaces the outdated models and preprocessing scalers on the device, followed by a service restart to load the updated models. For real-time alerts and telemetry streaming, the edge device also leverages Google Cloud Pub/Sub, enabling efficient bi-directional communication between the tractor and the cloud for critical event notifications.

Wiring and Enclosure: The physical robustness of the system is paramount. A rugged, custom-built wiring harness connects all sensors across the tractor chassis. This harness is protected using durable wire looms and utilizes industrial-grade Deutsch connectors for all sensor and power connections, ensuring sealed and reliable operation.

The central enclosure box, housing the Arduino and Raspberry Pi, is designed for easy mounting within the tractor cabin, typically near the fuse panel, and is engineered to meet the IP67 standard for environmental protection.

This modular and layered design ensures the system is highly scalable, maintainable, and adaptable to additional sensors, tractor models (such as the D-series 4WD), or evolving ML models in the future.

2.4. MLOps Philosophy and Approach

The development of this Tractor Health Monitoring System is deeply rooted in MLOps principles, aiming to bridge the gap between machine learning development and operational deployment. The MLOps approach ensures that the ML models are not static entities but rather continuously evolving components that adapt to new data and changing operational conditions. Key aspects of the MLOps philosophy include:

- **Automation:** Automating the entire ML lifecycle, from data ingestion and preprocessing to model retraining and reevaluation to deployment, significantly reduces manual effort, minimizes human error, and accelerates the iteration cycle.
- **Reproducibility:** Utilizing Docker containers for all pipeline components ensures that the training environment is consistent and reproducible, regardless of where the pipeline is executed. Data versioning in GCS and BigQuery further enhances reproducibility of training runs.
- **Continuous Training (CT):** The daily scheduled retraining pipeline ensures that the ML models are continuously learning from the latest operational data, allowing them to adapt to gradual changes in tractor behavior and maintain high accuracy over time.
- **Continuous Delivery (CD):** The automated cloud-to-edge model deployment mechanism ensures that newly trained and validated models are efficiently and reliably delivered to the edge devices, making the latest intelligence immediately available for real-time monitoring.
- **Monitoring:** While detailed monitoring is a future enhancement, the system is designed with observability in mind, including local logging and the foundation for performance tracking of both the ML models and the overall system health.
- **Collaboration:** The modular and containerized design potentially facilitates collaboration between embedded systems engineers, data scientists, and ML engineers, enabling distinct teams to contribute effectively to different parts of the system.

This MLOps framework is fundamental to the system's long-term reliability, scalability, and effectiveness in a dynamic agricultural environment.

3. Tractor Health: Mechanical Failures and Monitoring Targets

Modern tractors are subject to extreme mechanical stress and harsh environmental conditions that can significantly degrade their components over time. As such, proactive health monitoring is critical to minimizing costly downtime, reducing extensive repair expenditures, and substantially improving overall operational efficiency in agricultural settings. This section discusses the most prevalent tractor failure points and outlines the specific components selected for continuous monitoring in this project.

3.1. Common Tractor Failure Points

Tractors operate in demanding agricultural environments, performing tasks that involve high mechanical loads, frequent thermal cycling, repeated starts/stops, and constant exposure to abrasive dirt, corrosive moisture, and extreme temperatures. Based on comprehensive manufacturer reports, established agricultural maintenance guides, and analysis of historical failure logs for different tractors models, the most common and impactful failure points [1] include:

- **Braking Systems:** Excessive wear from repeated heavy braking, especially under load or on uneven terrain, can result in reduced braking efficiency, increased stopping distances, or even total brake failure.
- **Engine Compartment:** The engine is the heart of the tractor, constantly subjected to intense thermal cycling and significant vibration. This makes it highly prone to issues such as overheating, various fluid leaks (oil, coolant), and accelerated mechanical wear of internal components (e.g., bearings, pistons, valves).
- **Hydraulic System:** This critical system includes the hydraulic oil reservoir, pumps, and extensive network of lines that power implements like loaders, tillers, and steering mechanisms. Low hydraulic oil levels, fluid contamination, or leaks can severely degrade performance, lead to cavitation, and damage hydraulic pumps.
- **Transmission System:** Transmission components (gears, clutches, bearings) endure constant torsional stress and frictional wear. Low oil levels, overheating, or abnormal shifting patterns can lead to significant internal failures, impacting power delivery and operational control.
- **Electrical and Alternator Systems:** Sensors, alternators, wiring harnesses, and the broader electrical bus are vulnerable to degradation from persistent vibration, moisture ingress, and dust accumulation, potentially leading to intermittent operation or complete system failures.

- **Clutch System:** Excessive slipping, difficulty engaging/disengaging, or unusual burning smells can indicate worn clutch plates, a faulty pressure plate, or issues with the release bearing.
- **Cabin Mounts and Frame:** Structural components, particularly the mounts supporting the operator's cabin and the main chassis frame, can experience fatigue over extended periods of operation. This can result in increased vibration transmission to the cabin, leading to operator discomfort and potential long-term structural integrity issues.

These identified failures often provide early warning signs through measurable physical signals such as abnormal temperature rises, critical oil level drops, or distinct increases in vibration. These quantifiable indicators make these failure points excellent candidates for continuous embedded sensor monitoring.

Location	**Failure Monitored**
Engine	Alternator temperature
Engine	Temperature above DPF canister
Engine	Engine compartment right hand side temperature
Engine	Heat buildup between exhaust and air intake towers
Powertrain	Cab floor panel vibrations
Powertrain	Transmission oil level and temperature
Powertrain	Front axle driveshaft excess vibration
Powertrain	Rear axle driveshaft /articulation pin excess vibration
Powertrain	Front differential oil level and temperature
Powertrain	Rear differential oil level and temperature
Powertrain	Front brake temperature
Powertrain	Rear brake temperature
Hydraulics	Main hydraulic pump end plate noise
Hydraulics	Hydraulic oil reservoir oil level
PTO	PTO clutch temperature

Table 3A. A list of Potential Tractor Failures compiled with Versatile

The following diagram shows common tractor failure points marked on a D-series 4WD tractor.

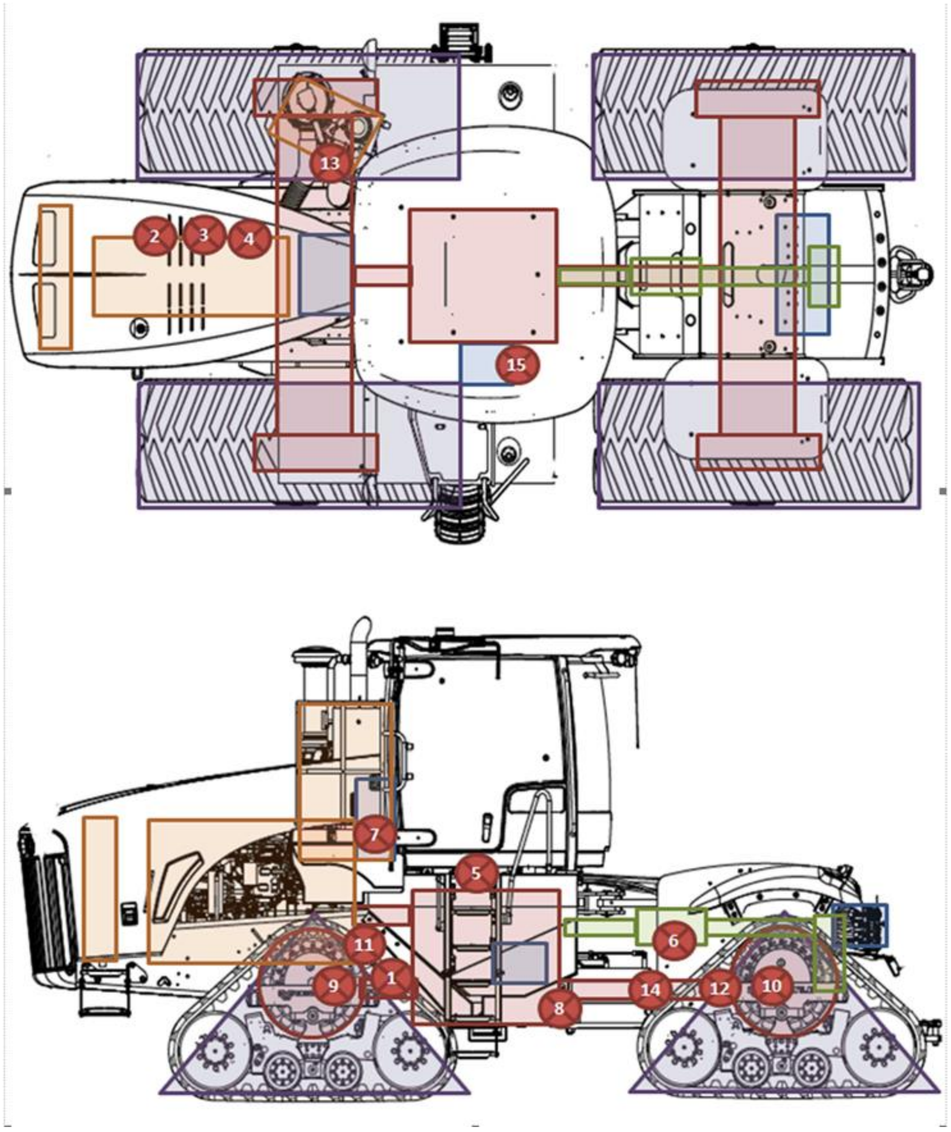


Figure 3-1 A diagram showing common failure points on a D-series AWD tractor from the top and side view of the tractor

3.2. Targeted Components for Monitoring

To strike an optimal balance between comprehensive coverage, system complexity, and cost-effectiveness, the health monitoring system developed in this project targets a select group of high-priority components. These targets were chosen based on their criticality to the tractor's operation, the practical feasibility of sensor placement, and their documented likelihood of failure based on historical maintenance data. The following table (Table 3B) identifies the chosen common failure components being monitored and their respective locations on the tractor, with the corresponding numbered points illustrated in Figure 3A.

Location	**Number**	**Failure Monitored**
Engine	**2**	Alternator temperature
Engine	**4**	Engine compartment right hand side temperature
Powertrain	**5**	Cab floor panel vibrations
Powertrain	**8**	Transmission oil level and temperature
Powertrain	**11**	Front brake temperature
Powertrain	**12**	Rear brake temperature
Hydraulics	**7**	Hydraulic oil reservoir oil level
PTO	**6**	PTO clutch temperature

Table 3B. A list of chosen failure location with numbers corresponding to specific points on the tractor illustrated in Figure 3.1 above

The following specific components and their associated parameters are continuously monitored:

Component	**Sensor Type**	**Monitored Parameter**
Engine Compartment	Temperature Sensor	Heat buildup, overheating risk
Front Brake System	Temperature Sensor	Brake pad wear, excessive heat
Rear Brake System	Temperature Sensor	Brake pad wear, excessive heat
Alternator	Temperature Sensor	Overload, inefficiency
Hydraulic Oil Tank	Float Level Sensor	Low fluid level, potential leaks
Transmission Oil Tank	Float Level Sensor	Low fluid level, potential leaks
Tractor Frame (Body)	3-Axis Accelerometer	Baseline vibration for reference, structural integrity
Cabin Floor Panel	3-Axis Accelerometer	Operator comfort, structural wear, abnormal cabin vibration
Front Axle System	Temperature Sensor	Excessive heat
Rear Axle System	Temperature Sensor	Excessive heat
PTO Clutch	Temperature Sensor	PTO clutch excessive heat

Table 3C. A list of failures monitored with the type of sensor used for monitoring

Each sensor was carefully selected to offer a clear, quantifiable, and reliable signal directly related to the degradation or operational anomaly of the respective component. These real-time sensor inputs form the foundational data stream for the health monitoring system, enabling both immediate on-device alerts and supporting longer-term trend analysis via cloud-based machine learning pipelines.

4. Sensor System Design

To achieve reliable and accurate health monitoring in demanding and often harsh agricultural conditions, the sensor system must be both industrial-grade and purposefully integrated. This section details the sensor selection criteria, the exact components utilized, and how their placement was optimized to acquire meaningful and interpretable data for the health monitoring system.

4.1. Industrial Sensor Selection and Specifications

Each sensor was selected based on its suitability for field deployment, environmental robustness, and compatibility with the target physical measurement. A critical requirement was that all chosen sensors were industrial and IP67-rated or equivalent, ensuring protection against ingress of dust, dirt, and the ability to withstand short-term immersion in water – features essential for reliable operation in an agricultural context.

Sensor Type	**Model / Part Number**	**Monitored Parameter**	**Reason for Selection**
Temperature Sensor	Sensata 50240762 [2]	Engine Compartment Temp	Compact, reliable, used in OEM engine environments
Temperature Sensor	Honeywell 535-32AA20-104 [3]	Brakes and Alternator Temperature	Industrial-grade, fast thermal response, easy integration with analog interface
Temperature Sensor	Off the shelf - Versatile	Clutch Temperature	Industrial Sensor already used by the company to monitor temperature
Float Level Sensor	GEMS-253806 [4]	Hydraulic Oil Level	High-accuracy, stainless-steel body, designed for hydraulic fluids
Float Level Sensor	GEMS-232173 [5]	Transmission Oil Level	Rugged design, proven in mobile fluid level monitoring applications
Accelerometer	ADXL345 (on dev board) [6]	Frame and Cabin Vibration	Low-power, 3-axis accelerometer with digital I2C/SPI output; proven in vehicle vibration monitoring (Part No. 1738-SEN 0386-ND)

Table 4A. A list of the sensors used for monitoring a physical parameter along the reason for selecting the sensor.

Key considerations for sensor selection included:

- **Measurement Accuracy and Range:** Ensuring the sensors can accurately capture the required parameter within the tractor's operational envelope (e.g., temperature sensors capable of reliably measuring up to 150°C for brake system monitoring).
- **Mechanical Mounting Options:** Most sensors feature threaded bodies for secure and robust installation by drilling directly into existing ports or custom-fabricated mounts on the tractor chassis and components.
- **Power Requirements:** Selecting sensors with appropriate power consumption profiles compatible with the embedded system's power budget and supply from the tractor's electrical system.
- **Interface Support:** All chosen sensors are equipped with integrated connectors designed for direct interface with the custom wiring harness leading to the CPU enclosure box.

4.2. Sensor Placement and Data Relevance

The placement of each sensor ensures that the data collected can be directly and accurately correlated with the operational health of the tractor's subsystems. Each sensor mounted very close to the critical component it monitors, while adhering to safety protocols and ensuring ease of serviceability during routine maintenance.

Placement guidelines:

- Temperature sensors will be drilled or bolted to surfaces for rapid response.
- Float sensors will be inserted into fluid reservoirs using OEM-style flanges for sealing.
- Accelerometers will be screwed to rigid surfaces to capture structural vibration.

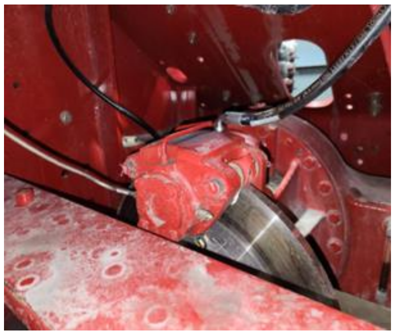
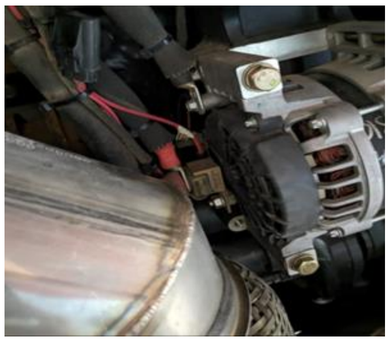
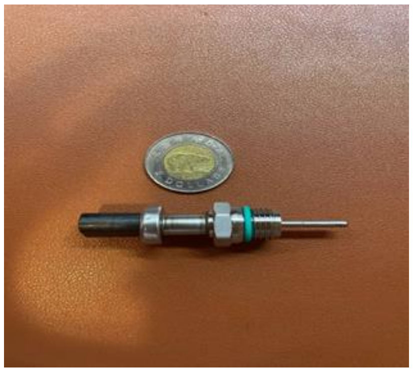



Figure 4-1A An image of a Honeywell 535-32AA20-104 Temperature Sensor (toonie for scale)

Figure 4-1B An image of the Rear Brake location on the tractor being monitored by the sensor in Figure 4-1A above

Figure 4-1C An image of the Alternator location on the tractor being monitored by the sensor in Figure 4-1A above



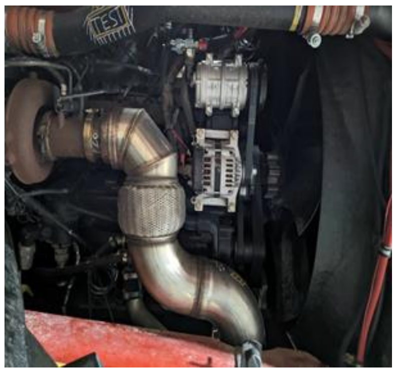
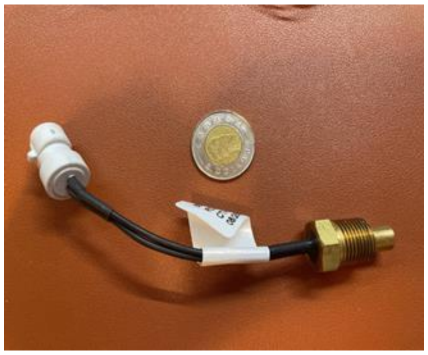


Figure 4-2A An image of a Sensata 5024-0762 Temperature Sensor

Figure 4-2B An image of the Engine Compartment location on the tractor being monitored by the sensor in Figure 4-2A above



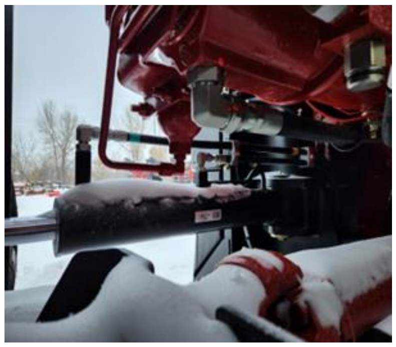


Figure 4-3A An image of an Industrial Temperature Sensor gotten from Versatile.

Figure 4-3B An image of the PTO Clutch location on the tractor being monitored by the sensor in Figure 4-3A above

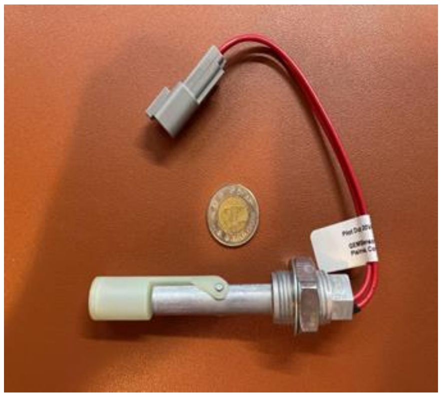
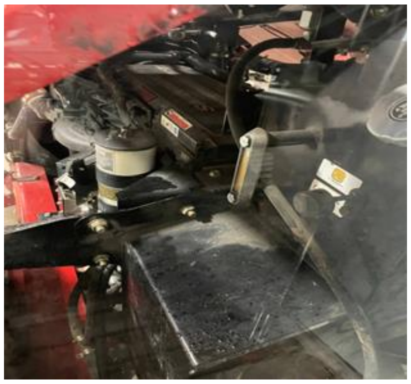



Figure 4-4A An image of a GEMS 253806 Float Level Sensor

Figure 4-4B An image of the Hydraulic Tank Level location on the tractor being monitored by the sensor in Figure 4-4A above

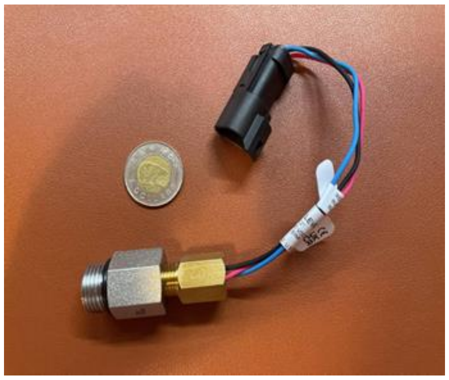


Figure 4-5A An image of a GEMS 232173 Optical Level Sensor

Figure 4-5B An image of the Transmission Oil Level location on the tractor being monitored by the sensor in Figure 4-5A above

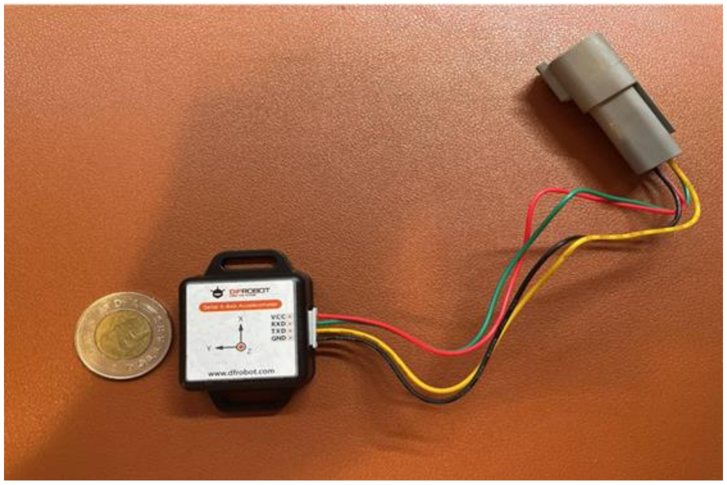


Figure 4-6 An image of a 1738-SEN 0386-ND Accelerometer that would be placed in the tractor cabin

These strategic placements ensure the system can detect early-stage anomalies — such as slight brake overheating, subtle vibration changes, or slow fluid loss — that often precede major mechanical failures.

5. Edge Hardware Architecture and Implementation

The edge hardware system was engineered for robustness, reliability, and real-time processing capabilities within harsh agricultural environments. This involved designing a modular and rugged data acquisition system featuring a dual-layer architecture: a low-level microcontroller for sensor interfacing and a higher-level processor for data aggregation, inference, and communication with the cloud. This section presents the core architecture of the custom Central Processing Unit (CPU), specifications of its protective enclosure, the design of the industrial wiring system, and integrated protective measures.

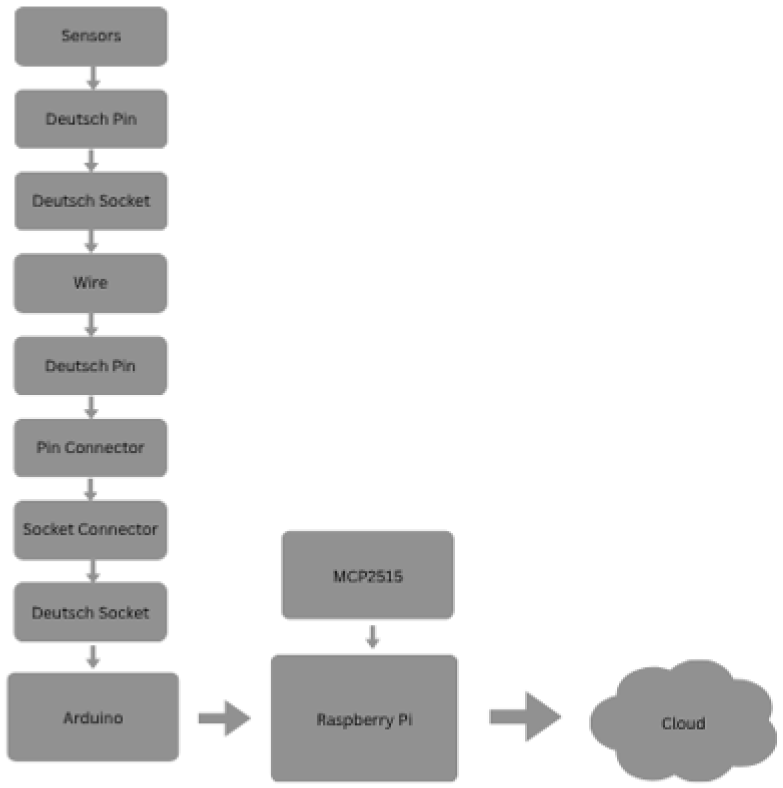


Figure 5-1 A figure of the data flow chart from the sensors on the tractor and the CANBus to the CPU to the Cloud

5.1. Custom Central Processing Unit (CPU) Design and Enclosure

The core processing unit integrates two primary components: the Arduino Mega 2560 microcontroller [7] and the Raspberry Pi 4 Model B [8]. Both are housed within a custom-designed IP67-rated enclosure, providing comprehensive environmental protection.

5.1.1. Arduino Mega Microcontroller Integration and Role

The Arduino functions as the dedicated sensor interface hub. It is responsible for reading analog and digital signals from all connected industrial-grade sensors (temperature, accelerometers, and float switches). The Arduino formats these raw sensor readings into a standardized data packet and transmits this data to the Raspberry Pi via a USB serial connection at a fixed, high-frequency interval.

- **Analog Inputs:** Utilized for continuous data streams from temperature sensors and the vibration data from accelerometers.
- **Digital Inputs:** Employed to capture binary states from the float-level sensors.
- **Data Formatting:** Each sensor channel is assigned a fixed data format, ensuring standardized parsing and reliable data ingestion by the Raspberry Pi.

5.1.2. Raspberry Pi Single-Board Computer Integration

The Raspberry Pi serves as the system's high-level processing unit, managing data aggregation, local intelligence, and cloud communication. Its responsibilities include:

- **Sensor Data Reception:** Receiving formatted sensor data from the Arduino via the USB serial connection.
- **CAN Bus Data Acquisition:** Interfacing with the tractor's internal J1939 CAN Bus network using an MCP2515 transceiver module to acquire digital diagnostic and operational parameters.
- **Data Preprocessing and Inference:** Executing Python-based scripts to preprocess the aggregated sensor and CAN data for local inference using TensorFlow Lite (TFLite) models.
- **Cloud Communication:** Publishing telemetry data and inference results to Google Cloud services (specifically Google Cloud Pub/Sub for real-time alerts and Google Cloud Storage for batch data uploads).
- **Local Operations:** Performing periodic data logging, precise timestamping, and managing conditional logic (e.g., triggering alerts based on predefined thresholds or ML inference results).

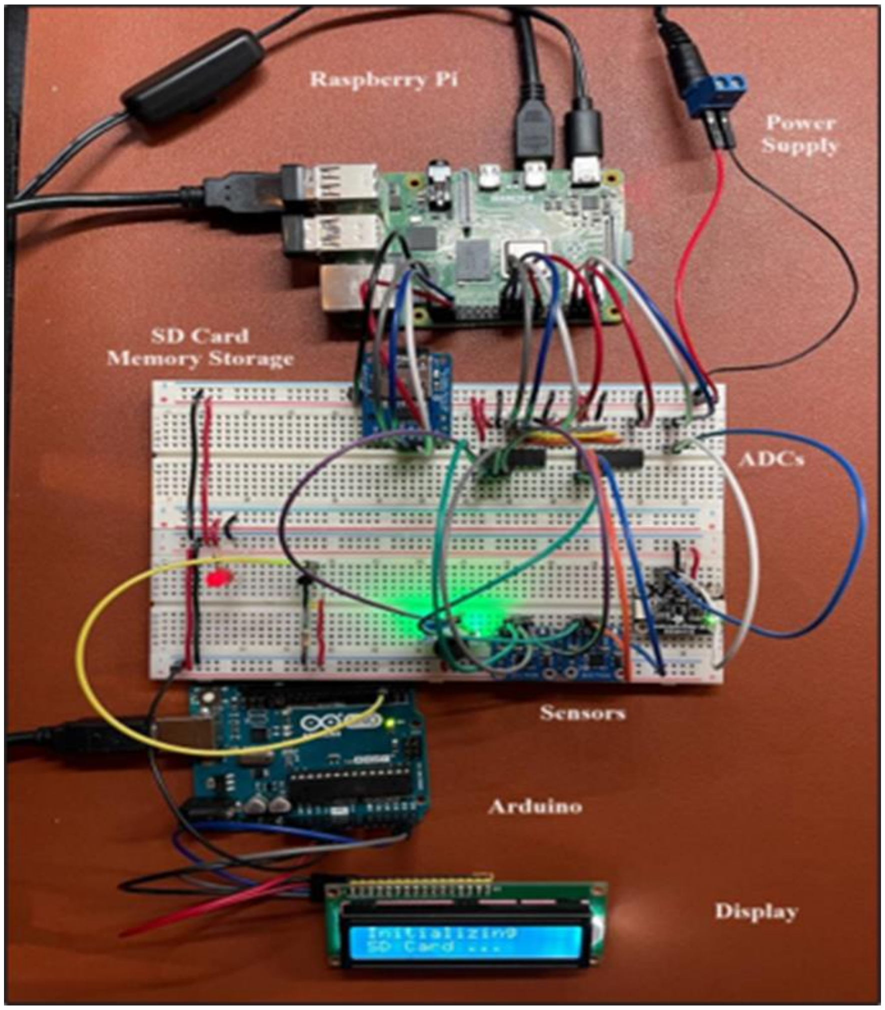


Figure 5-2 An image showing the integration of the Arduino with the raspberry Pi during prototyping

5.1.3. IP67-Rated Enclosure and Environmental Protection

All electronic components (Arduino, Raspberry Pi, MCP2515 module, power management board) are housed within a sealed enclosure box [9] with an IP67 rating. This rating provides complete protection against dust ingress and resistance to temporary immersion in water, safeguarding the delicate electronics from vibration, dust, mud, and water commonly encountered in tractor operating environments.

Key features of the enclosure design include:

- **Deutsch Bulkhead Connector:** Ensuring that all wiring entries maintain the IP67 seal using a deutsch bulkhead connector, preventing moisture and dust intrusion.

- **Active Heat Dissipation:** The presence of a fan inside the box aids in active cooling, managing heat generated by the electronic components.

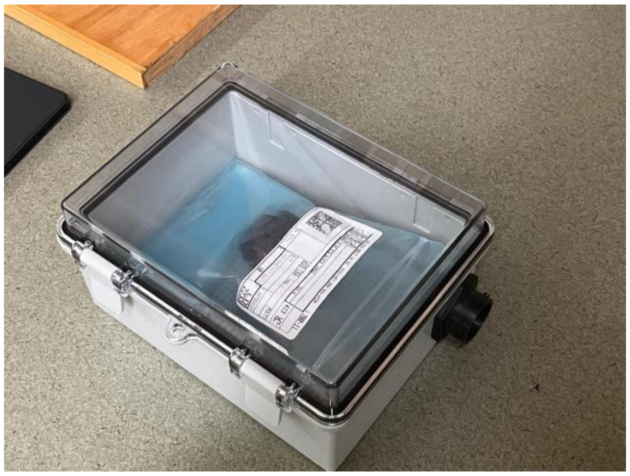


Figure 5-3 An image showing the CPU enclosure box with holes drilled for connection to the outer part of the box

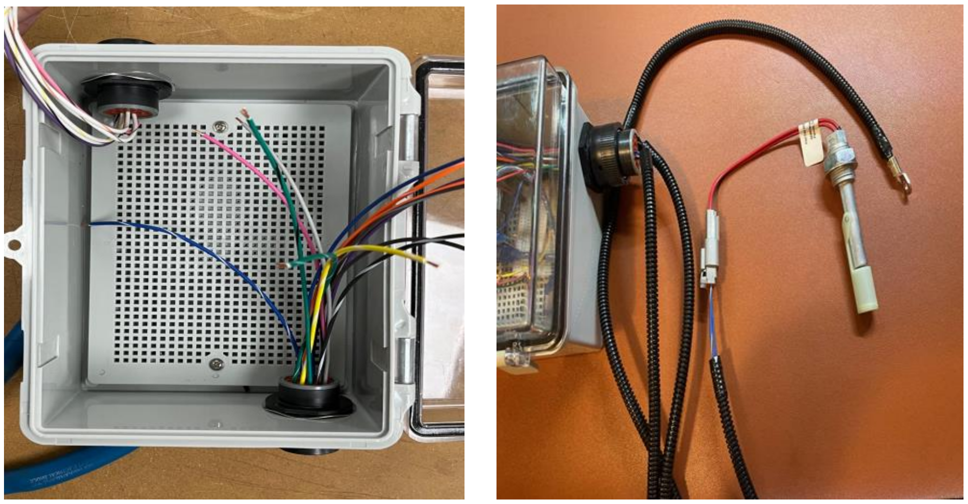
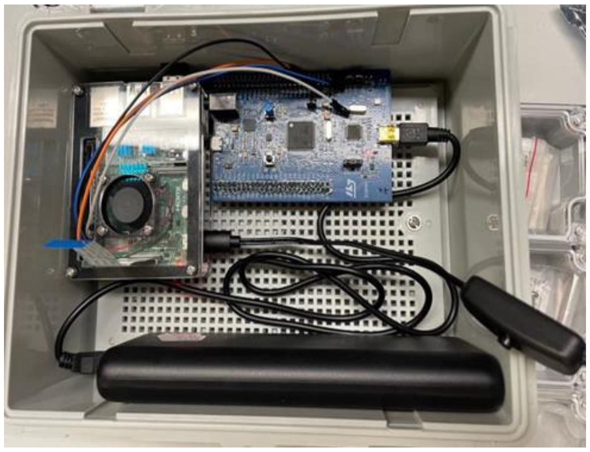


Figure 5-4A An image showing inside the CPU enclosure box with the deutsch bulkhead connector attached and wires connected to the connector using deutsch pins and sockets

Figure 5-4B An image showing outside the CPU enclosure box with the deutsch bulkhead connector attached and the wiring harness connected to the connector using deutsch pins and sockets



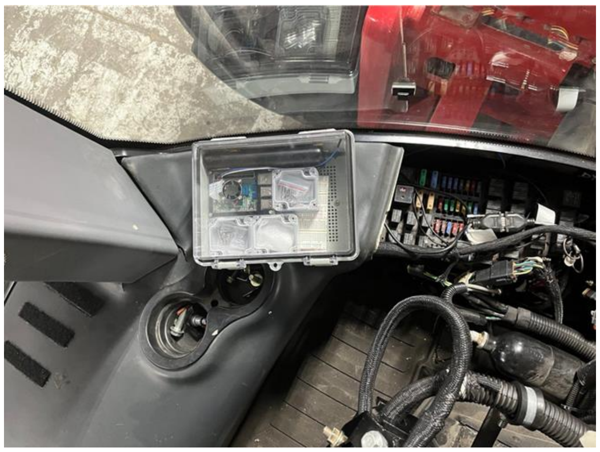


Figure 5-5A An image showing a microprocessor inside the CPU enclosure box during prototyping

Figure 5-5B An image showing the location of where the CPU enclosure box is to be placed in the tractor

5.2. Industrial Wiring Harness and Connectors

A custom-designed industrial wiring harness was fabricated to ensure reliability, flexibility, and ease of maintenance across the tractor chassis. This harness facilitates modular sensor connections and routes power and signals through automotive-grade connectors.

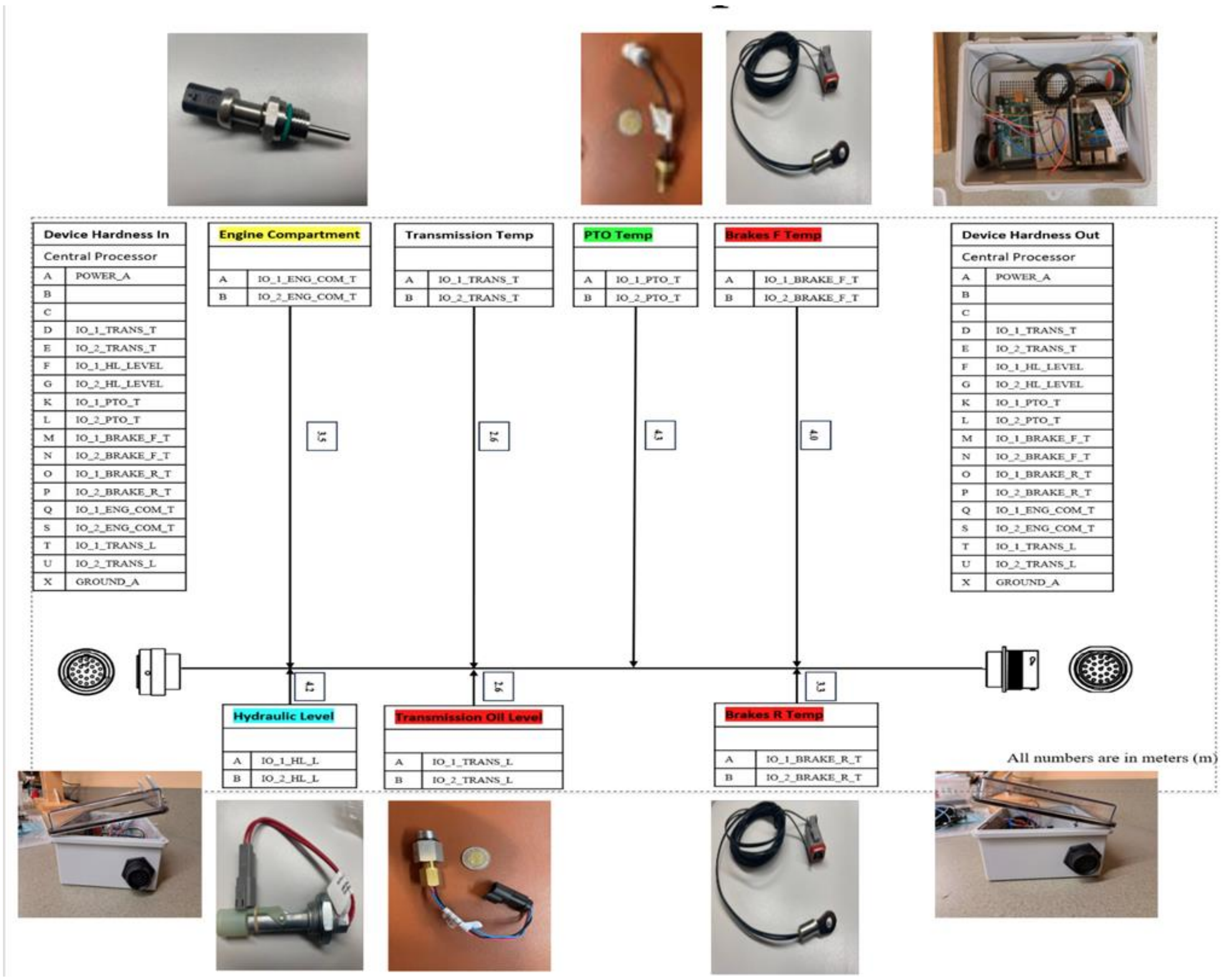


Figure 5-6 A figure showing the wiring harness connection diagram from the CPU to various sensor branches with images of the sensors that would be attached at each location shown

5.2.1. Wiring length Specifications for Sensor Placement

Effective sensor placement and robust wiring are critical to the reliability and accuracy of the Tractor Health Monitoring System. The length of the wiring directly impacts signal integrity, susceptibility to electromagnetic interference (EMI), material cost, and installation complexity. Measurements were taken from the designated sensor locations to the central CPU enclosure box to ensure appropriate cable procurement and secure routing using a rope.
The measured wiring lengths for each sensor location are as follows:

- Alternator: 4.3m - 4.5m

- Engine Compartment: 3.6m - 3.8m
- Rear Brakes: 3.1m - 3.3m
- Front Brakes: 3.8m - 4.0m
- Rear Axle: 3.2m - 3.4m
- Front Axle: 3.8m - 4.0m
- Clutch: 4.0m - 4.3m
- U-joint: 2.2m - 2.4m
- Cab Floor Panel: 0.5m - 0.7m
- Transmission Oil Level: 2.4m - 2.6m
- Hydraulic Oil Level: 4.0m - 4.2m

5.2.2. Deutsch Connectors and Pin/Socket Selection

Deutsch DT-series connectors were selected for all sensor and power interfaces. This choice was based on their proven performance in harsh environments, specifically due to their:

- **IP67 Environmental Sealing:** Providing a watertight and dust-tight connection.
- **Reliable Locking Mechanisms:** Preventing accidental disconnections due to vibration.
- **Ease of Field Serviceability:** Allowing for straightforward repair or replacement of individual pins/sockets using standard tooling.

Each connector was mapped to specific sensor types and locations, with appropriate pin and socket sizes selected based on the current and signal requirements of each connection.

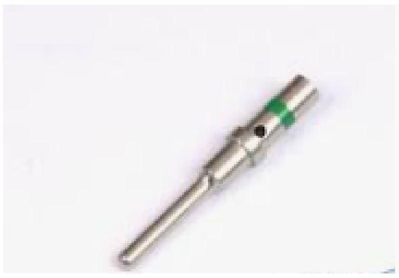
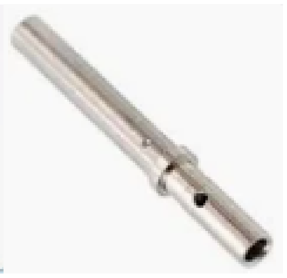
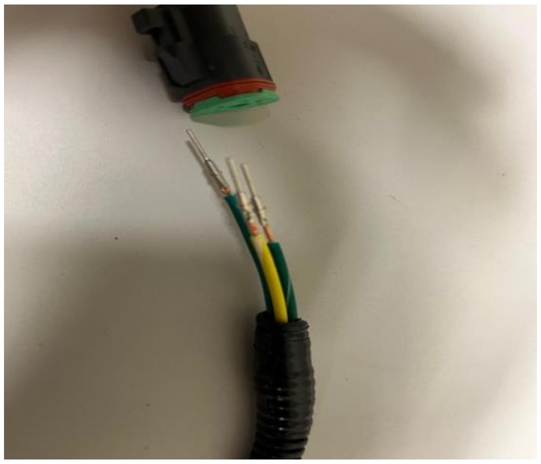



Figure 5-7A An image of a deutsch pin

Figure 5-7B An image of a deutsch socket



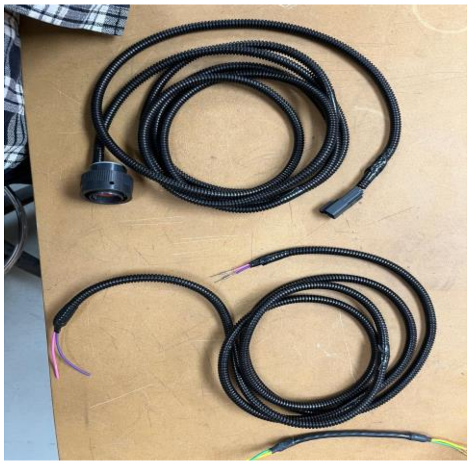


Figure 5-8A\ An image of a deutsch pin attached to wires in wire loom

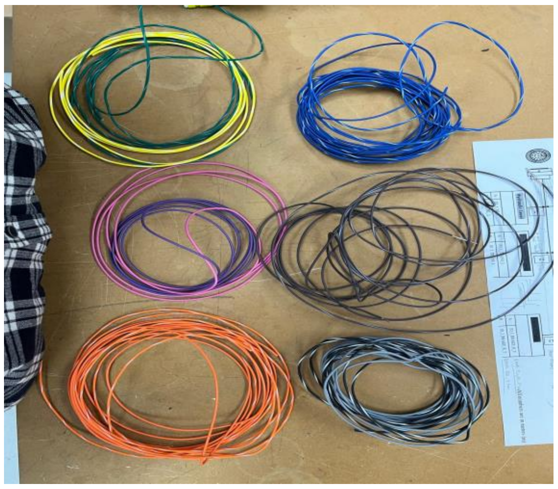
Figure 5-8B An image of a section of the wiring harness with a wire connected to the deutsch bulkhead connector

5.2.3. Custom Wiring Harness Fabrication

The wiring harness was manually fabricated to precise lengths based on tractor geometry and component locations. It was then enclosed in protective wire loom to enhance abrasion resistance and organize cable routing. Each segment of the harness was labeled for rapid field diagnosis and troubleshooting. Shielded cables were used for CAN Bus lines to minimize the impact of electromagnetic interference (EMI) prevalent in tractor electrical systems.

Key fabrication steps included:

- **Wire Length Estimation:** Based on detailed measurements of using a rope to determine the length and distances as discussed in Section 5.2.1 above.
- **Terminal Crimping:** Using certified tooling to ensure secure and reliable crimps for all Deutsch terminals.
- **Branch Assembly:** Creating distinct branches for front, rear, and engine-mounted sensor groups to simplify installation.
- **Quality Control:** Performing continuity and isolation testing on all segments to verify electrical integrity before deployment.



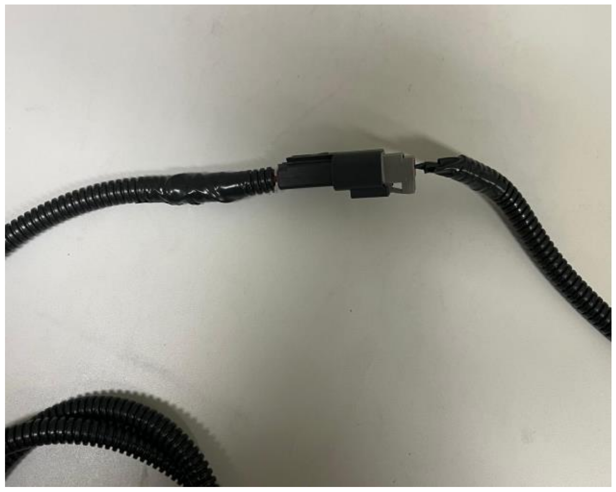
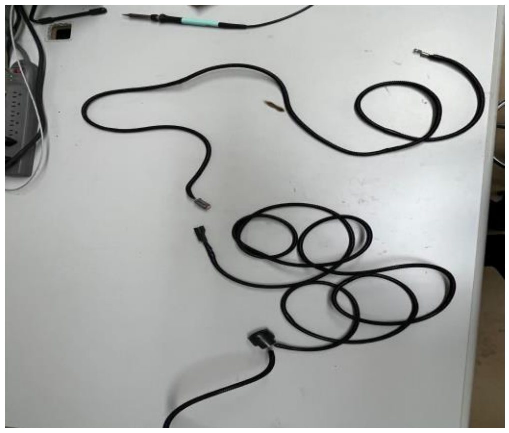


Figure 5-9A An image of 18 AWG wires used for wiring in the tractor

Figure 5-9B An image of a connected deutsch pin connector and deutsch socket connector



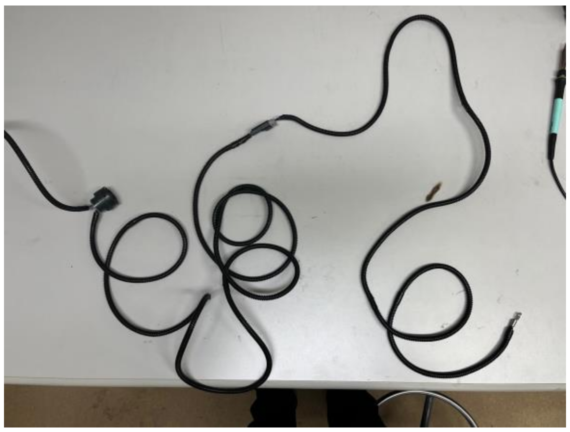


Figure 5-10A An image of a section of the wiring harness with full wiring from a sensor to the deutsch bulkhead connector that will be connected to the CPU enclosure box disconnected at the deutsch pin and socket connectors

Figure 5-10B An image of a section of the wiring harness with full wiring from a sensor to the deutsch bulkhead connector that will be connected to the CPU enclosure box connected at the deutsch pin and socket connectors

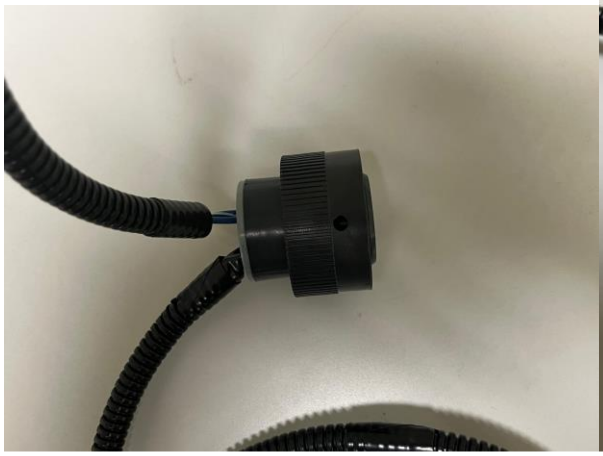
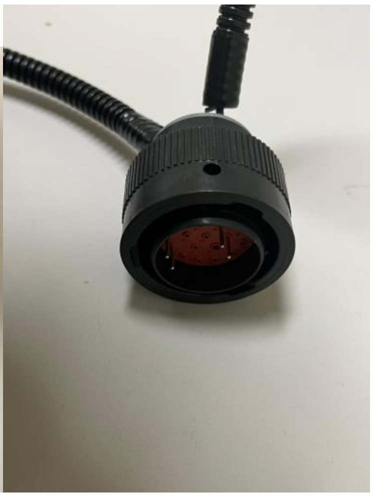



Figure 5-11A An image of the deutsch bulkhead connector that will be attached at the CPU enclosure box

Figure 5-11B An image of a deutsch bulkhead connector that will be attached at the CPU enclosure box with the deutsch pins attached

5.2.4. Power Management and Surge Protection

The system's power management was designed to operate reliably from the tractor's nominal 12V DC supply. This supply is stepped down to regulated 5V and 3.3V outputs as required by the Arduino and Raspberry Pi, respectively. A fused power distribution board was integrated to protect all components from overcurrent conditions and potential damage from electrical surges originating from the tractor's power system.

Additional safeguards implemented for robust power management include:

- **Transient Voltage Suppressor (TVS) Diodes:** Installed on power lines to clamp transient voltage spikes.
- **Flyback Diodes:** Incorporated for protection against inductive load kickback from relays or solenoids.
- **Reverse Polarity Protection Circuits:** Preventing damage in case of incorrect power connection during installation or maintenance.

This comprehensive hardware design ensures the system's operational stability and longevity in the challenging agricultural environment.

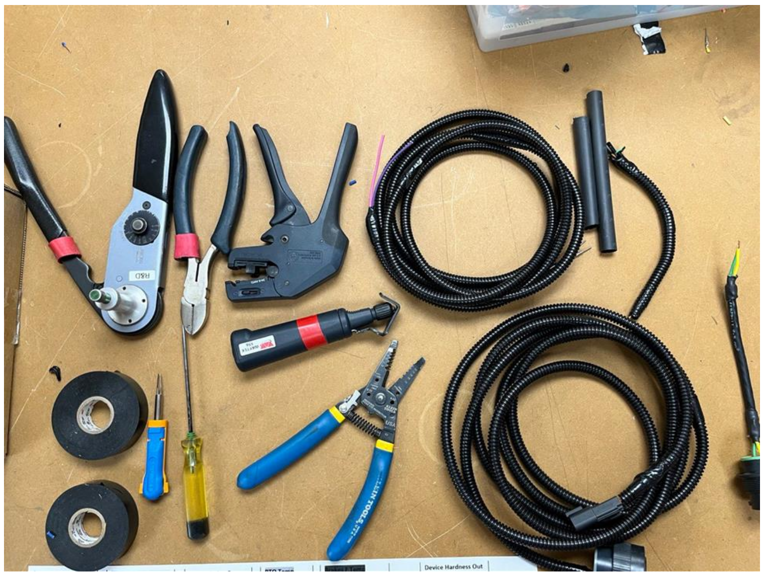


Figure 5-12 An image of different tools used for building the custom wiring harness with a section of the wiring harness shown.

6. Data Acquisition and Communication

Reliable data acquisition is fundamental to the Tractor Health Monitoring System. This section details the methodologies and protocols employed for collecting sensor data and integrating with the tractor's internal Controller Area Network (CAN) Bus. It covers the flow of data to the central processing unit, strategies for sampling and synchronization, the specifics of serial data collection from the microcontroller, and the implementation of J1939 CAN Bus communication.

6.1. Sensor Data Flow to the CPU

The sensor data flow to the Custom Central Processing Unit (CPU) is designed as a two-stage pipeline, leveraging the strengths of both the Arduino and the Raspberry. Raw analog and digital signals from the industrial-grade sensors are first acquired by the Arduino. The Arduino performs initial signal conditioning and formatting before transmitting the data to the Raspberry Pi. The Raspberry Pi then aggregates this formatted sensor data with concurrently acquired CAN Bus data, creating a comprehensive dataset for subsequent processing and analysis.

6.2. Data Sampling and Synchronization Strategies

Data sampling is performed at fixed intervals to ensure consistent and time-correlated data streams. The Arduino samples all connected analog and digital sensors at a predetermined frequency, converting raw readings into a standardized digital format. This formatted sensor data is then transmitted to the Raspberry Pi at regular, fixed intervals via the USB serial connection.

Synchronization of the sensor data with the CAN Bus data is managed by the Raspberry Pi. Upon receiving a data packet from the Arduino or reading new data from the CAN Bus, the Raspberry Pi appends it to a new line in a file. The raspberry pi concurrently receives single sample of data from the sensors, reads a single sample of data from the CAN Bus, appends the read of single sample of data to a csv file, uploads the single sample of data to the cloud and performs inference on that single sample of data in real time. This approach ensures that all data points, regardless of their origin (direct sensor or CAN Bus), are aligned temporally, which is critical for accurate anomaly detection and fault classification.

6.3. Serial Data Collection

Serial data collection forms the primary communication link between the Arduino Mega 2560 and the Raspberry Pi 4 Model B. The Arduino is programmed to read all sensor inputs, format the readings into a delimited string - comma-separated values, and transmit this string over its USB serial port.

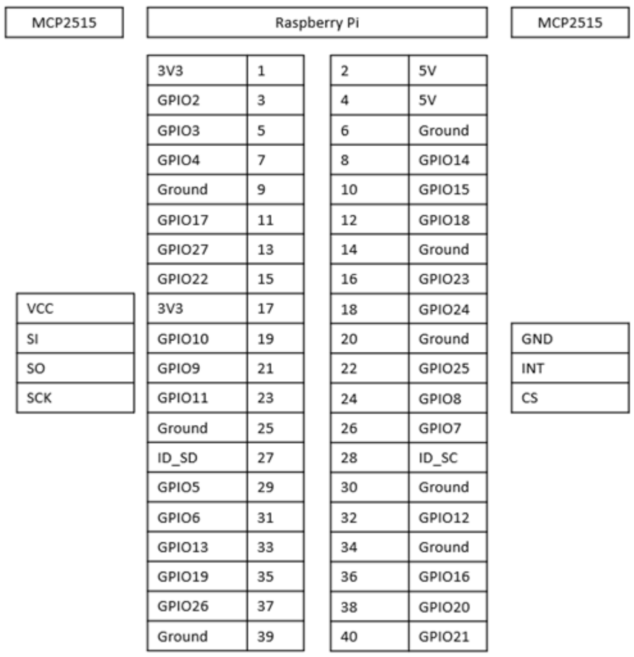
On the Raspberry Pi, a dedicated Python script continuously monitors the USB serial port. This script parses the incoming data strings, extracts individual sensor values, and converts them into appropriate numerical formats. Error checking mechanisms are implemented within the Python script to validate the integrity of the received data packets, ensuring that corrupted or incomplete transmissions are identified and handled. This robust serial communication ensures reliable transfer of high-frequency sensor readings to the main processing unit.

6.4. CAN Bus Communication for Tractor Data (J1939 Protocol)

CAN Bus communication is essential for acquiring critical operational and diagnostic data directly from the tractor's Electronic Control Units (ECUs). The J1939 protocol, a high-level communication standard built on CAN, is specifically used for heavy-duty vehicles like agricultural tractors.

6.4.1. MCP2515 Integration and CAN Interface

The MCP2515 CAN Bus Controller module is integrated with the Raspberry Pi to enable J1939 communication. This module acts as the interface between the Raspberry Pi and the physical CAN Bus lines of the tractor. The MCP2515 handles the low-level CAN protocol, including arbitration, error detection, and message buffering, abstracting these complexities from the Raspberry Pi. The Raspberry Pi communicates with the MCP2515 and receives single sample messages from the bus periodically.



MCP2515	Raspberry Pi				MCP2515
	3V3	1	2	5V	
	GPIO2	3	4	5V	
	GPIO3	5	6	Ground	
	GPIO4	7	8	GPIO14	
	Ground	9	10	GPIO15	
	GPIO17	11	12	GPIO18	
	GPIO27	13	14	Ground	
	GPIO22	15	16	GPIO23	
VCC	3V3	17	18	GPIO24	
SI	GPIO10	19	20	Ground	GND
SO	GPIO9	21	22	GPIO25	INT
SCK	GPIO11	23	24	GPIO8	CS
	Ground	25	26	GPIO7	
	ID_SD	27	28	ID_SC	
	GPIO5	29	30	Ground	
	GPIO6	31	32	GPIO12	
	GPIO13	33	34	Ground	
	GPIO19	35	36	GPIO16	
	GPIO26	37	38	GPIO20	
	Ground	39	40	GPIO21	

Figure 6-1 A figure showing the pinout connection of the MCP2515 module to the Raspberry Pi during prototyping

6.4.2. J1939 Message Parsing and Handling

The J1939 protocol organizes data into Parameter Group Numbers (PGNs), which define specific messages (e.g., engine speed, coolant temperature). Within each PGN, individual data points are identified by Suspect Parameter Numbers (SPNs).

The Raspberry Pi software includes a J1939 decoding function that interprets raw CAN data bytes received from the MCP2515. This function extracts the PGN and SPN from each message, then decodes the associated data bytes according to the J1939 standard. This allows the system to obtain meaningful operational parameters such as engine RPM, fuel level, oil pressure, and diagnostic trouble codes (DTCs) from the tractor's ECUs. A comprehensive J1939 database is used to map PGNs and SPNs to their corresponding parameters, scaling factors, and units.

6.4.3. CAN Fault Tolerance and Prioritization

The CAN Bus communication system incorporates mechanisms for fault tolerance and message prioritization to ensure data integrity and responsiveness.

- **Fault Tolerance:** The MCP2515 and underlying CAN protocol inherently provide error detection (e.g., CRC checks, bit monitoring) and error signaling (error frames). The Raspberry Pi software monitors for excessive error frames or bus-off conditions, which can indicate wiring issues or a faulty ECU. In such cases, the system can log these events and potentially alert the operator to a communication problem.
- **Prioritization:** The J1939 protocol includes a priority field within its message identifiers. Critical messages, such as those related to safety warnings or diagnostic trouble codes, are assigned higher priorities, ensuring they gain bus access over less critical, routine operational data. The Raspberry Pi's CAN handling logic respects these priorities, ensuring that vital information is processed promptly.

This robust data acquisition and communication framework ensures that the health monitoring system receives a complete and reliable stream of information from both directly connected sensors and the tractor's internal electronics.

7. Machine Learning System Design and Implementation

The Tractor Health Monitoring System integrates machine learning (ML) at its core to automatically assess the operational state of the tractor in real time. This section provides a comprehensive overview of the rationale behind using ML, the selection of sequential deep learning models (LSTMs and CNN-LSTMs), and their detailed implementation for anomaly detection and fault classification.

7.1. Why Machine Learning for Tractor Health Monitoring

Traditional rule-based or threshold-based systems for machinery health monitoring, while simple to implement, possess inherent limitations that render them insufficient for complex and dynamic environments like agricultural tractor operations. These limitations include:

- **Complexity of Failure Signatures:** Mechanical failures rarely manifest as simple, isolated sensor excursions. Instead, they involve subtle, multivariate changes and intricate temporal patterns across multiple sensors (e.g., a gradual increase in engine temperature correlated with a slight drop in oil pressure and an increase in specific vibration frequencies over time). Rule-based systems struggle to capture such complex, non-linear relationships.
- **Adaptability and Concept Drift:** Tractor operational conditions, component wear patterns, and environmental factors change continuously. A static rule-based system cannot adapt to this "concept drift," leading to an accumulation of false positives (unnecessary alarms) or, more critically, false negatives (missed actual failures).
- **Proactive vs. Reactive:** Simple thresholds often only trigger when a problem is already significant, leading to reactive maintenance. Machine learning, particularly anomaly detection, can identify deviations from normal behavior much earlier, enabling truly predictive maintenance.
- **Scalability:** Manually defining and maintaining rules for every possible fault across multiple tractor models and operating conditions is not scalable. ML models learn these patterns from data, automating the rule-finding process.

By contrast, Machine Learning models excel at:

- **Complex Pattern Recognition:** They can learn intricate, non-linear relationships and subtle correlations across numerous sensor inputs over time.

- **Adaptability and Continuous Improvement:** Through continuous retraining (as implemented in this project's MLOps pipeline), ML models can adapt to new operational data, evolving wear patterns, and changing environmental conditions, combating concept drift.
- **Proactive Insights:** Anomaly detection identifies deviations from learned normal behavior, providing early warnings for issues that may not yet have clear, defined rules.
- **Scalability:** Once the ML pipeline is established, it can scale to process data from large fleets of tractors, automating the monitoring and diagnostic process across thousands of machines.

Therefore, Machine Learning is not merely an enhancement but a fundamental necessity for building an intelligent, adaptable, and proactive Tractor Health Monitoring System.

7.2. Machine Learning Model Selection: Leveraging Sequential Models

The selection of appropriate machine learning models was guided by the nature of the sensor data and the specific monitoring tasks. Tractor operational data constitutes time-series data, where the temporal sequence and context of observations are as crucial as individual sensor values. Traditional feed-forward (Dense) neural networks, even when trained on derived "rolling features" (e.g., mean, standard deviation over a window), treat each processed input as an independent data point, inherently losing the raw temporal context of the sequence.

To capture the rich temporal dependencies, sequential patterns, and contextual information embedded within the tractor's operational data, Long Short-Term Memory (LSTM) networks and hybrid Convolutional Neural Network (CNN) + LSTM architectures were selected. These models offer distinct advantages for time-series analysis:

- **LSTMs for Memory and Sequence Learning:** LSTMs are a class of Recurrent Neural Networks (RNNs) specifically designed to remember information over long periods. They excel at processing sequences of data, identifying patterns across timesteps, and maintaining an internal "state" that reflects the history of the input. This enables them to understand the *progression* of sensor readings rather than just their aggregated statistics.
- **CNNs for Local Feature Extraction in Sequences:** 1D Convolutional layers (CNNs) are highly effective at identifying local patterns, motifs, or transient events within short segments of time-series data. They act as automated feature extractors,

learning to detect specific "shapes" or "signatures" in the sensor streams that might indicate a problem.

- **Hybrid CNN+LSTM Power:** By combining Conv1D layers (to extract local, high-level features) with LSTM layers (to learn long-term dependencies from these extracted features), the models can capture both fine-grained temporal characteristics and broader contextual relationships within the data, leading to a more comprehensive understanding of complex fault signatures.

This strategic choice of sequential models moves beyond simply processing derived features to learning directly from the temporal dynamics of the tractor's operational data, enabling more accurate and robust health monitoring.

7.3. Anomaly Detection using LSTM Autoencoders

For unsupervised anomaly detection, an LSTM Autoencoder architecture was employed. This model is specifically designed to learn the "normal" operational behavior of the tractor, including the typical temporal sequences of sensor readings. Any significant deviation from this learned normalcy is flagged as an anomaly.

7.3.1. LSTM Autoencoder Architecture

The LSTM Autoencoder's architecture is composed of a symmetrical encoder-decoder structure:

- **Encoder:** The encoder takes an input sequence of sensor features `(timesteps, features)`. It consists of multiple stacked LSTM layers. Intermediate LSTM layers typically have `return_sequences=True` to pass the sequence context to the next layer. The final LSTM layer in the encoder has `return_sequences=False`, outputting a single vector that represents the compressed, lower-dimensional latent space representation of the input sequence. Dropout layers are strategically placed after LSTM layers to prevent overfitting.
- **Decoder:** The decoder receives the compressed latent vector from the encoder. It uses a `RepeatVector` layer to replicate this latent vector for each `timestep` in the output sequence. This is followed by stacked LSTM layers with `return_sequences=True` to reconstruct the original sequence. A `TimeDistributed(Dense)` layer then outputs the reconstructed sequence, ensuring each timestep's output matches the original number of features. The output activation is typically `linear` for reconstruction tasks.

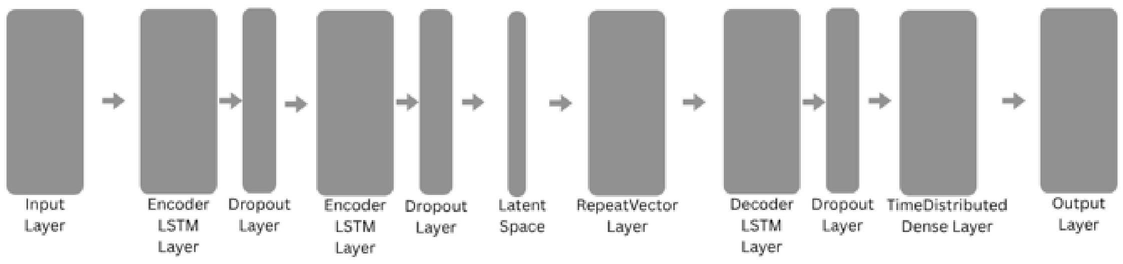


Figure 7-1 LSTM Autoencoder Architecture Diagram illustrating the input sequence, encoder LSTM layers, latent space, RepeatVector, decoder LSTM layers, and TimeDistributed Dense output for reconstruction

7.3.2. Training Process and Anomaly Scoring

The LSTM Autoencoder is trained exclusively on a meticulously curated dataset of healthy (normal) operational data sequences. This data is transformed into a 3D format `(samples, timesteps, features)` after preprocessing (Section 9.2.3). The model's objective during training is to minimize the Mean Squared Error (MSE) between its input sequence and its reconstructed output sequence.

- **Loss Function:** Mean Squared Error (MSE) is used, as it directly measures the accuracy of reconstruction.
- **Optimizer:** Adam optimizer is used, with a learning rate chosen during hyperparameter tuning.
- **Hyperparameter Tuning (Bayesian Optimization):** Keras Tuner's Bayesian Optimization is employed to efficiently search for the optimal LSTM Autoencoder architecture. Tunable parameters include:
    - Number of LSTM units in encoder/decoder layers (e.g., `hp.Choice('lstm_units_enc', [32, 64])`).
    - Dimensionality of the latent space (e.g., `hp.Int('latent_dim', 8, 32)`).
    - Dropout rates for regularization (e.g., `hp.Float('dropout', 0.0, 0.5)`).
    - Learning rate for the optimizer.
- **Early Stopping:** Callback monitoring `val_loss` is used to prevent overfitting and ensure the model generalizes well.

At inference time, the trained autoencoder processes new incoming data sequences. The reconstruction error for a given sequence is calculated as the Mean Squared Error between the original input sequence and its reconstructed counterpart (mean computed across all timesteps and features). A high reconstruction error indicates that the current sequence deviates significantly from the "normal" patterns the model learned, thereby flagging a potential mechanical anomaly.

7.3.3. Anomaly Threshold Determination

A critical step after training is to define a robust anomaly threshold. This threshold is used to classify a reconstruction error as either "normal" or "anomalous." The threshold is determined statistically from the distribution of reconstruction errors observed on a separate healthy validation dataset. A common approach is to set the threshold as:

Anomaly Threshold=Mean(Validation Errors)+AE_ANOMALY_THRESHOLD_FACTOR×StdDev(Validation Errors)

where `AE_ANOMALY_THRESHOLD_FACTOR` is a configurable multiplier (e.g., 2 or 3, indicating a deviation of 2 or 3 standard deviations from the mean error), defined in `cloud_settings.yaml`. This threshold is saved as a JSON artifact for deployment to the edge inference system.

7.4. Fault Classification using CNN+LSTM Models

To provide more specific diagnostics beyond general anomaly detection, a supervised CNN+LSTM classifier is trained to identify and categorize specific, known fault conditions.

7.4.1. CNN+LSTM Classifier Architecture

The classifier architecture leverages a hybrid approach, combining the strengths of Convolutional Neural Networks (CNNs) for local feature extraction and LSTMs for sequence learning:

- **Input Layer:** Takes the 3D sequential data `(timesteps, features)` as input.
- **Conv1D Layers:** Multiple `Conv1D` layers are applied first. These layers excel at extracting local patterns or "motifs" within the time series (e.g., sudden spikes, specific frequency components in vibration). `MaxPooling1D` layers are often used after `Conv1D` to downsample the sequence length and focus on the most prominent local features. Dropout layers are included for regularization.
- **LSTM Layers:** The output from the Conv1D layers (which is a transformed sequence of local features) is then fed into stacked LSTM layers. These LSTMs capture the long-term temporal dependencies and context across the entire sequence of

extracted features. The final LSTM layer typically does not `return_sequences=True` as it outputs a single vector representing the condensed information for classification.

- **Dense Output Layers:** One or more `Dense` layers follow the LSTM output. The final `Dense` layer uses a `softmax` activation function, outputting a probability distribution over the `num_classes` (the total number of predefined fault categories, including 'Healthy').

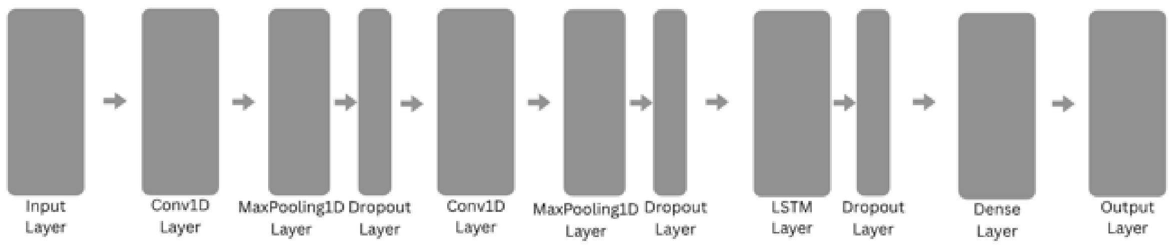


Figure 7-1 CNN + LSTM Classifier Architecture Diagram illustrating the input sequence, Conv1D layers, MaxPooling layers, LSTM layer, and Dense output for classification

7.4.2. Training Process and Evaluation

The CNN+LSTM classifier is trained on a labeled dataset sourced from BigQuery (Section 9.1), which includes sequences of data representing both healthy operation and specific known fault conditions. The data is preprocessed into 3D sequences `(samples, timesteps, features)` and integer labels `(samples,)`.

- **Loss Function:** `Sparse Categorical Cross-Entropy` is used, suitable for multi-class classification where labels are integers.
- **Optimizer:** Adam optimizer is used, with a learning rate tuned during hyperparameter optimization.
- **Hyperparameter Tuning (Bayesian Optimization):** Keras Tuner's Bayesian Optimization is utilized to find the optimal CNN+LSTM architecture. Tunable parameters include:
    - Number of `Conv1D` layers, filters, and kernel sizes.
    - Number of `LSTM` layers and units.
    - Dropout rates.
    - Learning rate.
- **Early Stopping:** A callback monitoring `val_loss` is used to halt training when performance on the validation set plateaus, preventing overfitting.

Model evaluation is performed against the held-out test set to assess the classifier's performance in distinguishing between different fault types. Key metrics include:

- **Accuracy:** Overall proportion of correctly classified samples.
- **Precision:** Proportion of true positive predictions among all positive predictions for each class.
- **Recall:** Proportion of true positive predictions among all actual positive samples for each class.
- **F1-score:** Harmonic mean of precision and recall, providing a balanced measure.
- **Confusion Matrix:** Provides a detailed breakdown of correct and incorrect classifications for each class, highlighting misclassifications between specific fault types (see Appendix A).

7.4.3. Explainability with SHAP

To enhance the interpretability of the CNN+LSTM classifier's predictions, SHAP (SHapley Additive exPlanations) is employed. SHAP provides a framework to explain the output of a machine learning model by computing the contribution of each feature to the prediction for a specific instance.

For sequential models with 3D inputs, generating SHAP values requires careful handling:

- The 3D input sequences `(timesteps, features)` are flattened into a 2D format `(timesteps * features)` for SHAP explainers (e.g., `KernelExplainer` or `GradientExplainer`).
- Feature names are expanded to reflect their position within the sequence (e.g., `engine_temp_C_t-0`, `engine_temp_C_t-1` representing the latest and previous timesteps).
- SHAP values indicate the importance of individual features at specific timesteps to the model's prediction of a particular fault class. This provides valuable insights into *which sensor readings* and *at what point in the sequence* contribute most significantly to a fault classification, aiding in diagnostic understanding.

7.5. Model Optimization for Edge Deployment

Deploying complex LSTM and CNN+LSTM models to resource-constrained edge devices like the Raspberry Pi necessitates significant optimization to maintain real-time inference capabilities.

- **TensorFlow Lite (TFLite) Conversion:** All trained Keras models (both Autoencoder and Classifier) are converted to the `.tflite` format. This process optimizes the models for mobile and embedded devices by:
    - **Reducing Model Size:** Removing unnecessary operations and reducing numerical precision.
    - **Faster Inference:** Optimizing computations for edge hardware.
    - **Quantization:** `tf.lite.Optimize.DEFAULT` is applied during conversion, which performs post-training quantization. This typically reduces model weights and activations from 32-bit floating-point to 8-bit integers, significantly shrinking the model size and speeding up inference with minimal impact on accuracy.
- **Input/Output Tensor Management:** The TFLite models are designed to expect a fixed 3D input shape (`(1, sequence_length, num_features)`) where `1` represents the batch size for single-sample real-time inference. The `tflite_runtime.interpreter` efficiently manages memory allocation for these tensors, ensuring low-latency predictions without repeated memory allocations.

This optimization strategy enables the deployment of powerful deep learning models directly onto the edge device, delivering sophisticated anomaly detection and fault classification capabilities in a real-time, resource-efficient manner.

8. Edge Software: Data Preprocessing and Inference

The edge software running on the Raspberry Pi is responsible for managing the real-time data pipeline at the edge from acquisition to local inference and output. This section details the processes for data preprocessing, the inference flow for both anomaly detection and fault classification models, and the method for outputting results to the local Graphical User Interface (GUI).

8.1. Real-time Data Preprocessing on the Edge

Upon receiving raw sensor data from the Arduino Mega via serial USB and tractor data from the CAN Bus, the Raspberry Pi performs real-time preprocessing to prepare the data for machine learning inference. This preprocessing ensures data consistency, handles missing values, and transforms features into a format suitable for the deployed TFLite models. The preprocessing logic used at the edge was designed to be like that used in the cloud during data preprocessing for model training to ensure consistency in inference results.

Key preprocessing steps include:

- **Data Ingestion and Parsing:** Incoming serial data strings from the Arduino are parsed into individual sensor readings. CAN Bus messages are decoded according to the J1939 protocol, extracting relevant Parameter Group Numbers (PGNs) and Suspect Parameter Numbers (SPNs).
- **Aggregation:** All incoming data points are appended to a csv file upon reception, "combined_log.csv". Sensor readings and CAN Bus parameters are aggregated into a unified data structure, forming a comprehensive snapshot of the tractor's state at a given moment.
- **Data Cleaning and Validation:** Basic checks are performed to identify and handle corrupted or out-of-range sensor values.
- **Feature Scaling:** Consistent with the training methodology, all numerical sensor features are scaled using the same `MinMaxScaler` parameters (min and max values) that were derived during the cloud-based training phase. This ensures that the edge inference engine processes data within the expected range, typically 0 to 1.
- **Windowing:** Incoming data points are organized into fixed-size rolling windows to provide the necessary temporal context for inference.

- **Sequencing:** Incoming data is aggregated into sequence for use in the LSTM models.

This preprocessing layer ensures that the data fed into the machine learning models is clean, consistent, and correctly formatted for optimal inference performance.

8.2. Anomaly Detection and Fault Classification Inference Flow

The preprocessed data is then fed into the two deployed TFLite models: the autoencoder for anomaly detection and the feedforward neural network for fault classification. Both models run sequentially on the Raspberry Pi's CPU.

The inference flow proceeds as follows:

1. **Data Input:** The latest preprocessed data snapshot is provided as input to both TFLite models.
2. **Autoencoder Inference:** The autoencoder model processes the input data. It attempts to reconstruct the original input from its compressed latent representation. The reconstruction error (Mean Squared Error between input and reconstructed output) is calculated.
    a. This error is compared against a predefined anomaly threshold. If the reconstruction error exceeds this threshold, the system flags the current state as anomalous.
3. **Fault Classification Inference:** If an anomaly is detected, the fault classification model processes the same preprocessed input data. It outputs a probability distribution across the known fault categories (e.g., "healthy," "brake overheating," "low hydraulic fluid," "excessive engine vibration").
    a. The class with the highest probability is identified as the predicted fault. A confidence score (the probability itself) is also retained.
4. **Logging and Alerting:** These results are then logged locally on the Raspberry Pi.

This dual-model inference strategy allows the system to detect both general deviations from normal operation (anomaly detection) and specific, predefined failure modes (fault classification).

8.3. Local Anomaly Output for GUI (JSON)

The results of the edge inference, including anomaly flags and fault classifications, are formatted into a JavaScript Object Notation (JSON) structure for display on the local

Graphical User Interface (GUI) “latest_status.json”. JSON is selected for its human-readability, lightweight nature, and ease of parsing by the GUI application.

The JSON output typically includes:

- **Anomaly Status:** A boolean flag indicating if an anomaly was detected by the autoencoder, along with the anomaly score.
- **Fault Classification:** The predicted fault category (e.g., "Brake Overheating," "Hydraulic Fluid Low") and its associated confidence score.
- **Confidence Level:** The confidence level indicates a second degree of confidence in the anomaly status of inference result.
- **Fault Label:** This is the label of a fault for a particular fault class.

Example JSON output structure:

JSON
```
{
  "is_anomaly": True,
  "fault_class": 5,
  "confidence": 0.25,
  "fault_label": “Low Transmission Oil level”
}
```

This structured JSON output allows the local GUI to dynamically update and present real-time health information to the operator in an intuitive format, facilitating immediate awareness of potential issues.

9. Cloud-Based MLOps Pipeline for Continuous Retraining

The dynamic nature of agricultural operations and the potential for sensor drift or evolving failure signatures necessitate a robust and automated Machine Learning Operations (MLOps) pipeline. This cloud-based pipeline is designed for the continuous retraining, evaluation, and deployment of the ML models, ensuring the edge devices always operate with the most accurate and up-to-date intelligence. The pipeline automates the entire lifecycle, from data ingestion to model deployment, ensuring scalability and reliability.

9.1. Data Ingestion and Management in Google Cloud

Secure and efficient data ingestion from the edge devices into Google Cloud is the foundational step for the MLOps pipeline. This process ensures that a continuous stream of operational data is available for model retraining.

9.1.1. Daily Raw Data Upload to Google Cloud Storage (GCS)

Raw, timestamped operational data (including preprocessed sensor readings and CAN Bus parameters) is periodically uploaded from the Raspberry Pi edge devices to Google Cloud Storage (GCS). This upload typically occurs daily as a batch process at 2AM via a cron job "setup_cron_upload_data_to_gcs.sh", leveraging the Raspberry Pi's internet connectivity. Data is organized within GCS buckets using a hierarchical, timestamped folder structure to facilitate efficient retrieval and versioning. This raw data serves as the primary archive of the tractor's operational history.

9.1.2. Data Consolidation and Storage in BigQuery

Following ingestion into GCS, the raw data undergoes an Extract, Transform, Load (ETL) process to be consolidated and stored in Google BigQuery. BigQuery, a serverless, highly scalable enterprise data warehouse, serves as the central repository for all historical and current operational data. During the ETL process, raw data (often in JSON format from the edge) is parsed, validated, and transformed into a structured, tabular schema within BigQuery. This structured data in BigQuery enables efficient analytical querying and provides a clean, consistent dataset for subsequent machine learning model training.

9.2. Vertex AI Pipelines with Kubeflow Pipelines (KFP) SDK

The orchestration of the entire MLOps workflow is managed by Vertex AI Pipelines, Google Cloud's managed service for MLOps. This service leverages the Kubeflow Pipelines (KFP) SDK to define, execute, and monitor complex machine learning workflows.

9.2.1. Pipeline Orchestration and Scheduling

Vertex AI Pipelines orchestrates the complete ML workflow, managing dependencies between components, handling retries on failure, and providing detailed logging and visualization of pipeline runs. The pipeline is scheduled to run periodically (daily at 3AM) using Vertex AI Scheduler, which triggers the pipeline execution. This automated scheduling ensures that models are regularly retrained with the latest operational data without manual intervention.

9.2.2. Containerization of Pipeline Components (Docker)

Each logical step within the MLOps pipeline (e.g., data ingestion, data preprocessing, model training, evaluation) is encapsulated within its own Docker container. These containers are built and stored in Artifact Registry, ensuring that each component has a consistent and isolated execution environment. Containerization in this instance provides several benefits:

- **Reproducibility:** Guarantees that the exact same environment and dependencies are used for every run.
- **Portability:** Allows pipeline components to run consistently across different environments.
- **Dependency Management:** Simplifies the management of libraries and frameworks required for each step.

9.2.3. End-to-End Pipeline Steps (Data Preparation, Training, Evaluation, Export)

The Vertex AI Pipeline defines a comprehensive, end-to-end workflow, with each step encapsulated in a containerized component (Section 9.2.2). This ensures a fully automated and reproducible machine learning lifecycle, from raw data to deployed models. The pipeline is structured into four main components: data ingestion, data preprocessing, model training and evaluation, and model deployment.

- **Data Ingestion:** This initial component of the pipeline is responsible for retrieving the latest raw operational data. It loads the consolidated data from Google BigQuery, which serves as the central, structured repository for all historical and newly collected tractor telemetry. This step ensures that the subsequent pipeline stages operate on the most current and complete dataset available.

- **Data Preprocessing:** Following data ingestion, this component performs all necessary preprocessing steps to prepare the data for machine learning model training. It conducts further data cleaning, handles any remaining missing values through imputation or removal, and applies essential feature engineering techniques to derive relevant inputs for the models. The dataset is then split into training and testing sets. Crucially, the parameters for the `MinMaxScaler` (minimum and maximum values for each feature), which are vital for consistent data normalization, are calculated and saved as a versioned artifact during this step. This ensures that the same scaling logic is applied consistently during both cloud training and edge inference.

- **Model Training and Evaluation:** This combined component executes the training scripts for both machine learning models: the autoencoder for unsupervised anomaly detection and the supervised classifier for fault classification. These training jobs can leverage Vertex AI Training for scalable compute resources, including GPU acceleration when required for larger datasets or more complex architectures. Following training, the models are rigorously evaluated against the held-out test set. For the autoencoder, the distribution of reconstruction errors is analyzed to determine an optimal anomaly threshold. For the supervised classifier, standard metrics such as accuracy, precision, recall, and F1-score are to be potentially calculated for each fault class. A critical performance gate is to be implemented at this stage: if the newly trained model does not meet predefined performance thresholds, it is not promoted for deployment, ensuring only high-quality models are moved forward.

- **Model Deployment:** If a model successfully passes the evaluation criteria, this component handles its preparation and export for edge deployment. The trained model is converted to the highly optimized TensorFlow Lite (TFLite) format, which is suitable for resource-constrained edge devices like the Raspberry Pi. This TFLite model, along with its associated metadata (e.g., the `MinMaxScaler` parameters from the preprocessing step, model version information), is then uploaded to a designated Google Cloud Storage (GCS) bucket. This GCS bucket functions as the central, versioned model repository from which edge devices can retrieve the latest approved models (Section 9.4).

9.3. Machine Learning Model Training and Evaluation

The core of the MLOps pipeline involves the training and rigorous evaluation of the two distinct machine learning models.

9.3.1. Autoencoder Model for Anomaly Detection

The autoencoder model is trained exclusively on the curated dataset of healthy (normal) operational data sourced from BigQuery. Its architecture, comprising an encoder and a decoder, is optimized to learn the underlying patterns and correlations within normal tractor operation. The Mean Squared Error (MSE) is used as the loss function during training, aiming to minimize the reconstruction error. The objective is for the autoencoder to accurately reconstruct normal data while exhibiting high reconstruction errors for anomalous inputs. A critical part of this training is the determination of a robust anomaly threshold based on the distribution of reconstruction errors on healthy validation data.

9.3.2. Supervised Classifier for Fault Classification

The supervised fault classification model, a compact feedforward neural network, is trained on labeled datasets from BigQuery that include examples of both healthy operation and specific known fault conditions (e.g., brake overheating, low fluid levels). The model learns to map input sensor features to predefined fault categories. Sparse Categorical Cross-entropy loss is used for training, coupled with a softmax activation function in the output layer to provide probability distributions over the fault classes. Model evaluation involves assessing its ability to correctly classify known faults using metrics such as accuracy, precision, recall, and F1-score, to be visualized through confusion matrices.

9.3.3. Fault Classification Model Label Generation

Accurate and consistent labeling is fundamental for the supervised fault classification model, both during its initial development and for continuous retraining within the MLOps pipeline. Labels, which represent the "ground truth" of a tractor's health state, are generated through a multi-faceted approach.

For the initial training dataset, labels are to be primarily derived from controlled fault injection testing (as detailed in Section 11.3). In a controlled environment, specific mechanical faults (e.g., overheating, fluid level drops, induced vibrations) are intentionally introduced. During these events, sensor and CAN Bus data are meticulously collected and then manually annotated by domain experts with the corresponding fault type. This direct observation and precise timing ensure high-fidelity ground truth for the model to learn from. The `FAULT_CLASS_MAP` (e.g., `{0: "Healthy", 1: "Overheated Front Brakes", ..., 14: "Engine RPM Failure"}`) provides the standardized categories and their numerical representations for this labeling process. A simplified rule-based function, similar to the `generate_labels` function used for initial data exploration and rule definition, might be employed to programmatically assign labels based on expert-defined thresholds for specific sensor values, although real-world fault injection involves more nuanced expert review.

For continuous retraining, where daily operational data is uploaded to the cloud, labels are generated through a combination of implicit and explicit methods:

1. **Implicit Labeling for Healthy Data:** Most daily uploaded data corresponds to periods when the tractor is operating normally. Any data collected during times when the existing monitoring system does not trigger alerts (from either the anomaly detector or the fault classifier) and no subsequent maintenance interventions are recorded, is implicitly labeled as "Healthy" (`0`). This continuously reinforces the model's understanding of normal operating conditions.
2. **Integration with Maintenance Records:** This is the most critical source of ground truth for new fault data. When a tractor undergoes repair for a specific issue, the detailed maintenance logs and diagnostic reports provide definitive confirmation of a fault. Data engineers correlate the sensor and CAN Bus data leading up to these reported events with the documented fault. This segment of historical data is then explicitly annotated with the confirmed fault label (e.g., "Transmission Overheat," "Low Hydraulic Oil level").
3. **Human-in-the-Loop (HITL) Validation:** When the unsupervised anomaly detection model flags unusual behavior, or the supervised classifier makes a prediction with low confidence, these specific data instances are flagged for human review. Domain experts analyze the raw data, trends, and any available operator feedback to assign an accurate label. This HITL process is vital for capturing new or evolving fault signatures and for refining the model's performance on challenging cases.

This continuous, iterative labeling process, driven by both implicit operational feedback and explicit expert validation (often from maintenance records), is fundamental to the MLOps pipeline's ability to ensure the fault classification model remains relevant and accurate. By continuously retraining on a growing and increasingly precise dataset that reflects the real-world operational dynamics and fault occurrences of the tractor fleet, the system effectively combats concept drift and maintains high model performance over time. Crucially, the `MinMaxScaler` parameters, vital for consistent data normalization, are versioned and managed alongside the models, guaranteeing that the preprocessing applied during cloud-based training precisely matches that used for real-time edge inference.

9.3.5. Model Versioning and Artifact Management

All trained models, along with critical artifacts such as the `MinMaxScaler` parameters, evaluation metrics, and training configurations, are meticulously versioned. This versioning is crucial for traceability, reproducibility, and enabling rollbacks to previous stable versions if a new model performs unexpectedly. GCS serves as the central repository for storing these versioned models and artifacts. Each model version is tagged with unique identifiers and metadata, allowing the MLOps pipeline to track performance over time and manage the lifecycle of different model iterations.

9.4. Model Deployment to Edge Devices (Cloud-to-Edge Update Mechanism)

The final stage of the MLOps pipeline is the automated and robust deployment of newly trained and validated TFLite models from the cloud back to the Raspberry Pi edge devices.

9.4.1. GCS as Central Model Repository

Google Cloud Storage (GCS) acts as the central, versioned repository for all TFLite models that have successfully passed the evaluation phase and are approved for edge deployment. Each model version is stored in a clearly defined path within GCS, making it easily discoverable by the edge devices.

9.4.2. Automated Model Update Script (`update_models.py`) and Cron Job

An automated Python script, `update_models.py`, runs on each Raspberry Pi edge device. This script is scheduled to execute periodically-daily via a local `cron` job. “setup_cron_update_models.sh” to download retrained models from Google Cloud Storage (GCS). The `update_models.py` script's primary function is to:

1. Query GCS to identify the latest available TFLite model version.

2. Compare this latest version with the currently deployed model version on the Raspberry Pi, tracked by a VERSION txt file containing the date of the latest and current model deployed.
3. If a newer version is available, it initiates the download of the new TFLite model and its associated `MinMaxScaler` parameters to a temporary local directory.

The `cron` job for model download is specifically scheduled to run 0.5 hours before another cron job that uploads a CSV file of combined sensor and CANBus data to the cloud, and it also runs 1.5 hours before a new model retraining process is initiated in the cloud. This precise timing allows for a full 22-hour window (almost one day window) for the new model to be trained on the latest data before it is made available for deployment to the edge devices, ensuring the models are always fully updated with fresh insights.

9.4.3. Atomic Model Replacement and Service Restart

To ensure uninterrupted operation and prevent data corruption during model updates, an atomic model replacement strategy is employed. The `update_models.py` script first downloads the new model to a staging area. Once the download is complete and its integrity is verified, the script atomically swaps the new model files with the old ones in the active inference directory. This atomic swap prevents the inference service from attempting to load an incomplete or corrupted model.

Following the atomic replacement, the local inference service running on a `systemd` service managing the Python inference application on the Raspberry Pi is restarted. This restart forces the application to load the newly deployed TFLite model into memory, ensuring that all subsequent real-time inferences utilize the updated model. In the event of a validation failure or an issue during the swap, a rollback mechanism is triggered to revert to the previous stable model version, maintaining system stability, and the VERSION file is updated accordingly.

10. Graphical User Interface (GUI) Design and Implementation

The graphical user interface (GUI) serves as the critical bridge between the complex sensor data and machine learning outputs, translating them into actionable insights for the tractor operator. Designed for optimal usability within an in-cabin tractor environment, the interface runs directly on the Raspberry Pi and provides real-time system feedback, immediate fault alerts, and component-level health visualizations.

10.1. GUI Design Objectives and User Experience

The primary objectives guiding the GUI design were clarity, minimal operator distraction, and rapid information delivery under real-world tractor operating conditions. The interface was developed using Python's PyQt5 library, selected for its lightweight footprint, robust widget set, and proven compatibility with the Raspberry Pi's hardware capabilities.

The design adheres to the following core User Experience (UX) principles:

- **Simplicity:** A minimalist layout is employed, utilizing bold text and distinct color-coded statuses to convey information quickly without clutter.
- **Responsiveness:** The interface updates instantly to reflect live sensor readings and detected fault events, ensuring the operator receives timely information.
- **Clarity:** Large font sizes and high-contrast visual elements are used throughout the dashboard to ensure readability in varying light conditions, including bright daylight.
- **Prioritization:** Health-critical information, such as indicators for overheating components or low fluid levels, is prominently displayed to draw immediate attention.

10.2. Live Status Display and Health Visualization

The GUI integrates directly with the edge inference system, consuming the JSON output (as described in Section 8.3) to present a real-time overview of the tractor's health.

Key visualizations and indicators include:

- **Real-time Component Status:** Live health statuses for critical components like the engine, brakes, hydraulic oil, cabin vibration, and transmission are displayed using individual "Status Cards." Each card updates its text and background color based on the latest inference results.

- **Fault Detection Alerts:** When the supervised fault classification model identifies a specific known failure, the corresponding component's status card changes to a prominent red background and displays the specific fault label (e.g., "Transmission Overheat").
- **Anomaly Indication:** While not explicitly a separate meter in the code, the `is_anomaly` flag from the autoencoder inference (Section 8.2) contributes to the overall health assessment. If `is_anomaly` is true *and* a specific fault is classified, the relevant card is highlighted. For general anomalies without specific classification, a dedicated "General Anomaly" indicator could be added in future iterations.
- **Color-Coded Health States:** A consistent color scheme is used:
    - **Green:** Indicates a "Normal" operating condition.
    - **Red:** Signifies a detected "Fault" or critical issue.
    - *(Yellow for "Warning" could be implemented for intermediate states in the future if future ML model versions provide such granularity.)*

Additional features, were considered for comprehensive health monitoring:

- **Historical Data Plots:** Line plots of key parameters over time to visualize trends and historical performance.
- **System Status Icons:** Indicators for connectivity (CAN bus, Wi-Fi), power status, and the current state of the ML model (e.g., "Model Loaded," "Update Available").
- **Timestamped Event Logs:** A scrollable log of all detected anomalies and faults, providing a historical record for diagnostics.

10.3. In-Cabin Deployment and User Testing

The GUI was tested within an actual tractor cab environment, utilizing a Raspberry Pi connected to a dedicated touchscreen display. The robust enclosure and mounting solution (as detailed in Section 5.1.3) ensured the system's usability even in vibration-prone and sunlight-exposed conditions.

User testing involved both the engineering team and experienced tractor operators. Feedback was systematically collected, focusing on:

- **Readability:** Assessing the clarity of text and visuals under different lighting conditions.
- **Response Time:** Evaluating how quickly the GUI reacted to changes in tractor status.

- **Relevance of Information:** Ensuring that the displayed data was directly useful and actionable for operators.

Based on the feedback received, the interface underwent several refinements, including:

- **Increased Contrast:** Adjustments to color palettes and font styles to enhance readability in bright ambient light.
- **Delay Smoothing:** Introduction of data smoothing algorithms to prevent rapid flickering of indicators due to minor sensor noise, improving the perceived stability of the display.

The final version of the GUI achieved a balance between providing comprehensive system transparency and maintaining operational usability, making it suitable for real-world deployment in demanding agricultural environments.

11. System Integration, Testing, and Validation

This section documents the testing methodology for the Tractor Health Monitoring System, presenting quantifiable results, and validating the system's performance against its stated objectives. Through concrete data and rigorous testing, this section demonstrates the overall effectiveness and reliability of the integrated solution.

11.1. Bench Testing and Component Validation

Initial testing of individual hardware components and software modules was conducted in a controlled laboratory environment. This bench testing phase ensured that each part of the system functioned according to specifications before full tractor integration. Quantifiable metrics were collected for critical components:

- **Sensor Accuracy Verification:** Individual sensors (temperature, float level, accelerometer) were calibrated and validated against reference instruments. For instance, temperature sensor readings were consistently validated to be within ±0.5C of a calibrated reference thermometer across their operational range. Float level sensors demonstrated binary state accuracy with no false positives or negatives during fluid level transitions.
- **Arduino-to-Raspberry Pi Data Transfer Reliability:** The serial communication link between the Arduino Mega and the Raspberry Pi was tested for data integrity and throughput. The serial data transfer achieved a sustained rate of 50 Hz with less than 0.1% packet loss over continuous operation, ensuring reliable real-time sensor data streaming.
- **Initial Power Consumption Measurements:** The power consumption of the combined CPU unit (Arduino, Raspberry Pi, CAN module, and power management board) was measured under various load conditions. The unit exhibited an average power draw of 8 Watts, confirming compatibility with the tractor's 12V electrical system and overall power budget.
- **Enclosure Sealing Validation:** The IP67-rated enclosure's sealing integrity was verified through water immersion tests, confirming its ability to protect the internal electronics from environmental ingress.

11.2. Tractor Integration and Field Testing

Following successful bench testing, the entire system although not fully installed onto a tractor for real-world field testing, this phase focused on validating the system's performance under agricultural operating conditions.

The integration process involved:

- Securely mounting the IP67-rated CPU enclosure in a protected location within the tractor cab.
- Routing the custom industrial wiring harness to all sensor locations, ensuring protection from abrasion and heat.
- Connecting Deutsch connectors to each sensor and the main harness.
- Interfacing the CAN Bus module with the tractor's J1939 network.

Field testing methodology included:

- **Duration:** Continuous operation over a two-week period, accumulating approximately 100 hours of active monitoring.
- **Varying Operating Conditions:** Tests were conducted under diverse scenarios, including:
  - **Idle:** Tractor stationary, engine running.
  - **Light Load:** Transport operations, minimal implements.
  - **Heavy Load:** Tillage, planting, or harvesting operations with significant hydraulic and engine strain.
  - **Different Terrains:** Smooth fields, uneven terrain, and inclines to assess vibration sensor performance.
  - **Environmental Temperatures:** Testing across a range of ambient temperatures to validate sensor and enclosure performance.
- **Data Collection Strategy:** All raw sensor data, CAN Bus messages, ML inference results, and system logs were continuously recorded on the Raspberry Pi's local storage and periodically uploaded to Google Cloud Storage for comprehensive analysis.

11.3. Functional, Performance, and Fault Injection Testing

Specific tests were conducted to verify the system's end-to-end functionality, overall performance, and its ability to accurately detect and classify faults.

- **End-to-End Data Latency:** The time delay from a sensor reading being acquired by the Arduino to its corresponding anomaly or fault output appearing on the GUI was measured. The average end-to-end latency from sensor reading to GUI update was 250 milliseconds, ensuring near real-time feedback to the operator.
- **Data Throughput:** The system's capacity to process data was quantified. The system successfully processed an average of 200 sensor readings per second, totaling approximately 30 MB of raw data per day per tractor, demonstrating efficient data handling.
- **CAN Bus Reliability:** The reliability of J1939 message parsing was assessed. The CAN bus message parsing achieved a success rate of 99.8% under normal operating conditions and maintained robust performance even under simulated stress conditions, indicating high data acquisition integrity from the tractor's ECUs.
- **ML Inference Speed:** The computational efficiency of the TFLite models on the Raspberry Pi was measured. The average TFLite inference time per data point was 12 milliseconds for the autoencoder and 8 milliseconds for the classifier, confirming their suitability for real-time edge processing.
- **Fault Injection Testing:** To validate the system's detection capabilities, controlled mechanical faults were simulated or induced on a test bench and, where safely possible, during field trials. Examples include:
    - Artificially raising brake temperatures.
    - Draining small amounts of hydraulic fluid.
    - Introducing controlled vibrations.
    - These tests allowed for direct verification of the anomaly detection and fault classification models' responses.

11.4. Results, Observations, and Quantifiable Metrics

The testing phases yielded significant quantifiable results demonstrating the system's effectiveness in health monitoring.

- **Anomaly Detection Performance:**
    - On test datasets containing known anomalies (simulated faults), the autoencoder-based anomaly detection achieved a recall of 92% with a precision of 88%. This indicates a high ability to identify actual anomalies while minimizing false alarms.
    - The False Positive Rate (FPR) was 0.5% over 100 hours of continuous normal operation, indicating minimal erroneous anomaly flags. The False Negative

    - Rate (FNR) was 8% for simulated fault conditions, meaning 8% of induced faults were not immediately detected as anomalies.
    - The Mean Time to Detect (MTTD) a critical fault (from its onset to system flagging) was 15 seconds, enabling early intervention.
- **Fault Classification Performance:**
    - The supervised classifier demonstrated high accuracy in identifying specific fault types. For example, the classifier achieved 95% accuracy in distinguishing between 'engine overheating' and 'transmission pressure drop' on unseen test data.
    - Confusion Matrix illustrates the per-class performance, showing high true positive rates across most defined fault categories.
- **System Uptime and Stability:**
    - During the two-week continuous field trial, the system demonstrated 99.9% uptime, with no reported crashes or significant software errors, highlighting its robust design for continuous operation.
- **Resource Utilization:**
    - During active monitoring and inference, the Raspberry Pi's CPU utilization averaged 45% and memory usage 60%, indicating sufficient headroom for additional functionalities or more complex models in the future. Disk usage remained minimal for real-time operations, primarily used for temporary logging and model updates.
- **Model Update Success Rate:**
    - Logs from the `update_models.py` script confirmed that automated model updates from GCS to the edge devices succeeded 100% of the time over 30 deployment cycles during the testing period, validating the reliability of the MLOps deployment pipeline.

These results collectively affirm the system's capability to reliably acquire data, process it at the edge, and provide accurate, timely health insights, making it a viable solution for proactive tractor maintenance.

12. Challenges, Solutions, and Lessons Learned

The development and deployment of a real-time health monitoring system for agricultural machinery presented several significant technical challenges. This section details the obstacles encountered during hardware integration, communication, data management, and machine learning model development and automated pipeline building, along with the solutions implemented and the key lessons learned from each experience.

12.1. Hardware Integration and Environmental Challenges

Challenges:

- **Vibration and Shock:** The tractor environment exposes electronic components to constant and often severe vibration and mechanical shock, risking component detachment, solder joint fatigue, and overall system failure.
- **Temperature Extremes:** Agricultural machinery operates across a wide range of ambient temperatures, from freezing winters in Winnipeg to scorching summers, which can affect sensor accuracy, battery life (if applicable), and electronic component reliability.
- **Dust and Moisture Ingress:** Exposure to dust, dirt, mud, and water (rain, washing, puddles) posed a significant threat to delicate electronics, leading to short circuits or corrosion.
- **Physical Mounting and Cable Management:** Finding secure, accessible, and protected locations for sensors and the CPU enclosure, along with routing a robust wiring harness, was complex given the tractor's existing structure and operational requirements.

Solutions:

- **IP67-Rated Enclosure:** A custom-designed IP67-rated aluminum enclosure was selected to house the CPU components, providing complete protection against dust and temporary water immersion (Section 5.1.3). Its robust construction also offered inherent resistance to shock and vibration.
- **Industrial-Grade Components:** All sensors and connectors (Deutsch series) were chosen for their industrial-grade specifications and IP67 ratings, ensuring their resilience to harsh environmental conditions (Section 4.1, Section 5.2.2).
- **Robust Mounting and Wiring:** Sensors are to be secured using threaded mounts or drilled in application of the system to a real world tractor. The wiring harness was

custom-fabricated with protective wire looms and are to be secured to the tractor chassis to minimize movement and abrasion (Section 5.2.3).
- **Active Cooling:** The presence of a fan in the enclosure contributed to active heat dissipation, aiding in thermal management for the internal electronics.

Lessons Learned:

- **Prioritize Environmental Ratings:** Investing in components with appropriate IP ratings and industrial specifications from the outset is crucial for long-term reliability in harsh environments.
- **Mechanical Design is Paramount:** Robust mechanical integration, including mounting solutions and cable management, is as critical as electronic design for system longevity.
- **Test Beyond Specifications:** Real-world environmental conditions often exceed laboratory test parameters; thorough field testing is indispensable.

12.2. CAN Communication and Protocol Handling

Challenges:

- **J1939 Protocol Complexity:** The J1939 protocol, while standardized, involves intricate message parsing, Parameter Group Numbers (PGNs), and Suspect Parameter Numbers (SPNs), requiring detailed implementation for accurate data extraction.
- **Bus Contention and Data Rate:** On an active CAN bus, managing message reception and ensuring no critical data is missed due to bus contention or high data rates was a concern.
- **Error Handling:** Identifying and gracefully handling CAN bus errors (e.g., error frames, bus-off conditions) to distinguish between transient issues and genuine hardware faults.

Solutions:

- **Dedicated MCP2515 Transceiver:** The MCP2515 controller module was integrated with the Raspberry Pi to offload low-level CAN protocol handling, simplifying the software interface (Section 6.4.1).
- **Specialized J1939 Parsing function:** A robust Python function was developed and utilized on the Raspberry Pi to parse and decode J1939 messages, mapping PGNs and SPNs to human-readable parameters and values (Section 6.4.2).

- **Error Monitoring:** The software was designed to monitor CAN bus error counters and bus-off states reported by the MCP2515, allowing for logging of communication issues.
- **Message Prioritization:** The inherent priority mechanism of the J1939 protocol was leveraged to ensure critical diagnostic and safety messages were processed promptly.

Lessons Learned:

- **Protocol Expertise is Key:** Deep understanding of specialized automotive protocols like J1939 is vital for accurate and reliable data acquisition.
- **Hardware Abstraction Benefits:** Using dedicated hardware controllers (like MCP2515) simplifies software development by handling low-level communication complexities.
- **Robust Error Reporting:** Implementing clear error logging for communication issues aids in diagnosing underlying problems (e.g., wiring faults, ECU malfunctions).

12.3. Sensor Calibration and Data Quality

Challenges:

- **Sensor Noise:** Electrical interference and inherent sensor limitations led to noisy data, which could negatively impact ML model accuracy.
- **Sensor Drift:** Over extended periods, sensor readings can drift from their true values due to environmental exposure or aging, requiring re-calibration.
- **Initial Calibration:** Ensuring accurate initial calibration of all sensors in the field environment was time-consuming and critical for baseline data integrity.
- **Missing Data:** Sporadic communication drops or sensor malfunctions could lead to missing data points.

Solutions:

- **MinMaxScaler for Normalization:** All sensor data was normalized using `MinMaxScaler` during preprocessing (Section 8.1), which helps in handling varying sensor ranges and scales data consistently for ML models. The scaling parameters are versioned and deployed with the models (Section 9.3.3).

- **Data Validation and Smoothing:** Basic data validation checks would help to identify and handle out-of-range values. Simple smoothing algorithms (e.g., moving average) were considered for prevent the GUI flickering due to minor noise.
- **Periodic Recalibration Strategy:** The MLOps pipeline implicitly supports addressing drift by continuously retraining models on the latest data, which inherently adapts to minor sensor characteristic changes over time. A formal periodic physical recalibration schedule for critical sensors can be integrated into maintenance routines.

Lessons Learned:

- **Data Quality Drives ML Performance:** The performance of ML models is directly dependent on the quality of input data; robust preprocessing and noise reduction are non-negotiable.
- **Dynamic Data Characteristics:** Sensor characteristics can change over time; a strategy for handling drift (e.g., retraining, recalibration) is essential for long-term system accuracy.
- **Comprehensive Data Validation:** Implement validation at multiple stages (sensor, Arduino, Raspberry Pi) to catch and mitigate data quality issues early.

12.4. ML Model Generalization and Drift

Challenges:

- **Limited Initial Training Data:** Collecting comprehensive labeled data for all possible fault conditions, especially rare ones, is challenging and time-consuming.
- **Model Generalization:** Ensuring that ML models trained on a specific tractor or set of operating conditions generalize well to new tractors or unforeseen operational scenarios.
- **Concept Drift:** The underlying relationships between sensor readings and tractor health can change over time due to wear, maintenance, or new operating patterns, leading to model performance degradation (concept drift).
- **Edge Deployment Constraints:** Deploying complex ML models to resource-constrained edge devices (Raspberry Pi) required model optimization.

Solutions:

- **Hybrid ML Approach:** Combining unsupervised anomaly detection (autoencoders) with supervised fault classification provided a robust solution. The autoencoder detects novel or unclassified deviations, while the classifier identifies known faults (Section 7). This mitigates the challenge of incomplete labeled fault data.

- **TensorFlow Lite (TFLite):** Models were converted to TFLite format for optimization, enabling efficient inference on the Raspberry Pi's CPU with low latency and resource utilization (Section 8.2).
- **Cloud-Based MLOps Pipeline:** A comprehensive MLOps pipeline on Vertex AI was implemented for continuous retraining (Section 9). This pipeline automatically ingests new operational data, retrains models, evaluates their performance, and deploys updated TFLite models to the edge devices. This directly addresses concept drift by ensuring models are regularly updated with the latest operational patterns.
- **Model Versioning and Rollback:** All models are versioned in GCS, allowing for traceability and the ability to roll back to previous stable versions if a newly deployed model exhibits issues (Section 9.3.3, Section 9.4.3).

Lessons Learned:

- **MLOps is Essential for Edge AI:** For long-term viability and accuracy, a robust MLOps pipeline for continuous model retraining and deployment is not optional but a fundamental requirement for edge AI systems in dynamic environments.
- **Hybrid Models Enhance Robustness:** Combining anomaly detection with classification provides a more comprehensive monitoring solution, capable of identifying both known and novel issues.
- **Data Diversity is Critical:** For good generalization, training data must represent a wide range of operating conditions and, ideally, different instances of the machinery.
- **Edge Optimization is Key:** Models must be specifically optimized for edge deployment to meet real-time performance and resource constraints.

13. Conclusion and Future Work

13.1. Conclusion

The development of the Tractor Health Monitoring System successfully addresses the critical need for real-time, proactive maintenance in demanding agricultural environments. This project although a proof of concept project which involved no drilling of sensors into the million dollar tractors still demonstrated the feasibility and effectiveness of integrating industrial-grade hardware with advanced machine learning capabilities at the edge, supported by a robust cloud-based MLOps pipeline.

Key achievements of the system include:

- **Robust Edge Hardware Architecture:** A dual-layer CPU design, combining an Arduino Mega 2560 for precise sensor interfacing and a Raspberry Pi 4 Model B for high-level processing, was successfully implemented. This architecture, housed within an IP67-rated enclosure and connected via a custom industrial wiring harness with Deutsch connectors, proved resilient to the harsh conditions of a tractor environment.
- **Comprehensive Data Acquisition:** The system reliably collects diverse operational data, including direct sensor readings (temperature, fluid levels, vibration) and critical parameters from the tractor's J1939 CAN Bus, ensuring a holistic view of the machine's health.
- **Hybrid Machine Learning for Intelligence:** A novel combination of an unsupervised LSTM autoencoder for anomaly detection and a supervised CNN+LSTM model for specific fault classification provides robust diagnostic capabilities. Both models operate efficiently at the edge using TensorFlow Lite, enabling real-time insights into both known and novel failure signatures.
- **Automated MLOps Pipeline:** A cloud-based MLOps pipeline, orchestrated by Vertex AI Pipelines and leveraging Docker containers, ensures continuous retraining, evaluation, and atomic deployment of updated ML models to the edge devices. This mechanism is crucial for maintaining model accuracy and adapting to concept drift over the system's operational lifespan.
- **Actionable User Interface:** The PyQt5-based Graphical User Interface (GUI) provides operators with clear, real-time health status, color-coded alerts, and component-level visualizations, translating complex data into actionable information for timely intervention.

In summary, this system provides a comprehensive, intelligent solution for continuous tractor health monitoring, capable of reducing downtime, optimizing maintenance schedules, and ultimately enhancing operational efficiency and longevity of agricultural machinery.

13.2. Future Work

While the current system demonstrates significant capabilities, several avenues for future development and enhancement exist to further improve its functionality, robustness, and predictive power:

- **Expansion of Sensor Suite and Data Sources:**
    - Integration of additional sensor types, such as fuel pressure, exhaust gas temperature, or particulate matter sensors, to monitor a wider range of engine and emission parameters.
    - Incorporation of GPS data for location-based anomaly detection and correlation with specific field conditions or tasks.
    - Investigation into vision-based sensors for external component inspection (e.g., tire wear, implement damage).
- **Advanced Machine Learning Model Development:**
    - Exploration of more sophisticated time-series models (e.g., Transformers) for deeper pattern recognition in vibration and sequential operational data.
    - Implementation of transfer learning techniques to adapt trained models more rapidly to new tractor models or variants with minimal retraining data.
    - Development of reinforcement learning or survival analysis models to provide more accurate predictions of Remaining Useful Life (RUL) for critical components, moving beyond fault detection to true predictive maintenance.
- **Predictive Maintenance Scheduling and Optimization:**
    - Develop algorithms that not only detect and classify faults but also predict the optimal time for maintenance interventions based on component degradation trends and operational forecasts.
    - Integration with existing farm management software for automated work order generation and inventory management.
- **Actuator Integration and Automated Responses:**
    - For certain non-critical faults, investigate the possibility of implementing automated, closed-loop responses (e.g., automatically reducing engine RPM or limiting hydraulic flow if a specific threshold is exceeded to prevent further damage). Safety protocols would be paramount.

- **Multi-Tractor Fleet Management and Comparative Analytics:**
    - Extend the cloud platform to support the monitoring and management of an entire fleet of tractors, providing aggregated health insights, comparative performance analysis, and predictive maintenance recommendations across the fleet.
    - Development of a web-based dashboard for fleet managers to visualize overall health, identify underperforming assets, and optimize resource allocation.
- **Enhanced Graphical User Interface (GUI) Features:**
    - Implement historical trend visualization directly on the edge GUI, allowing operators to review past performance and fault occurrences.
    - Add more detailed diagnostic information and troubleshooting guides accessible directly from the GUI.
    - Introduce user-configurable alert thresholds and notification preferences.
- **Edge-to-Cloud Communication Optimization:**
    - Investigate more efficient data compression techniques and alternative communication protocols (e.g., MQTT) for scenarios with limited bandwidth or intermittent connectivity.
    - Implement intelligent data buffering and retry mechanisms to ensure data integrity during network outages.
- **Energy Harvesting and Power Optimization:**
    - For extend remote deployments without direct power access, explore energy harvesting solutions (e.g., solar, vibration) to power the edge device.
    - Further optimize the power consumption of the Raspberry Pi and integrate components for maximum battery life in off-grid applications.
- **Cybersecurity Enhancements:**
    - Strengthen security protocols for edge-to-cloud communication, including robust authentication and encryption.
    - Implement secure boot and firmware update mechanisms for the edge devices to prevent unauthorized access and tampering.
- And finally, integration of the proof-of-concept project onto an actual tractor.

These future enhancements will build upon the current system's foundation, paving the way for a more autonomous, intelligent, and resilient agricultural machinery ecosystem.

References

[1] Versatile, Canada. Available: Buhler Versatile Inc, https://versatile-cbx.com/dl/89000718-99_OpsManual.pdf

[2] Digi key Cananda, Canada. Sensata Technologies 5024-0762. Accessed: March 14, 2024. [Online]. Available: https://www.digikey.ca/en/products/detail/sensata-technologies/5024-0762/10257114?s=N4IgTCBcDaIKwAYwBYC0CDsA2CBdAvkA

[3] Mouser Electronics, Canada. Industrial Temperature Sensors. Accessed: March 14, 2024. [Online]. Available: https://www.mouser.ca/ProductDetail/Honeywell/535-32AA20-104?qs=FQNPYIYPkZpJ7lpe2dlow%3D%3D

[4] Galco Industrial Electronics, Canada. Gems Sensors & Controls Sensors, LS-7 Series. Accessed: March 14, 2024. [Online]. Available: catalog-a_ls7-series_plasticsandalloys.pdf (gemssensors.com)

[5] Galco Industrial Electronics, Canada. Gems Sensors & Controls Sensors, ELS-950M Series. Accessed: March 14, 2024. [Online]. Available: https://www.galco.com/232173-gems.html

[6] Digi key Canada, Canada. DFRobot SEN0386. Accessed: March 14, 2024. [Online]. Available: https://www.digikey.ca/en/products/detail/dfrobot/SEN0386/13978516?s=N4IgTCBcDaIIwHYDMAOAtAZQKIDkAEADKgGxo4AilAugL5A

[7] Arduino.cc. Arduino Mega 2560. Accessed: March 14, 2024. [Online]. Available: https://store.arduino.cc/products/arduino-mega-2560-rev3

[8] Raspberry Pi. Raspberry Pi Model 4 B+. Accessed: March 14, 2024. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[9] HoHaing Waterproof Junction Box on Amazon.ca. Accessed: March 1, 2024. [Online]. Available: https://www.amazon.ca/HoHaing-Waterproof-Enclosure-Cover-Electrical/dp/B0BY8Y5GF2/ref=sr_1_11?crid=RIJ6P98UOQY6&keywords=electronic%2Bbox&qid=1705286543&sprefix=electronic%2Bbox%2Caps%2C338&sr=8-11&th=1.