RE4DY

MANUFACTURING DATA NETWORKS

RE4DY TOOLKIT

Name of the Tool	FEDMA – Federated Maintenance for Milling Machines
Tool Owner	Industry Commons Foundation
Version	1.0
Date	Nov 2025
Version	V1.0

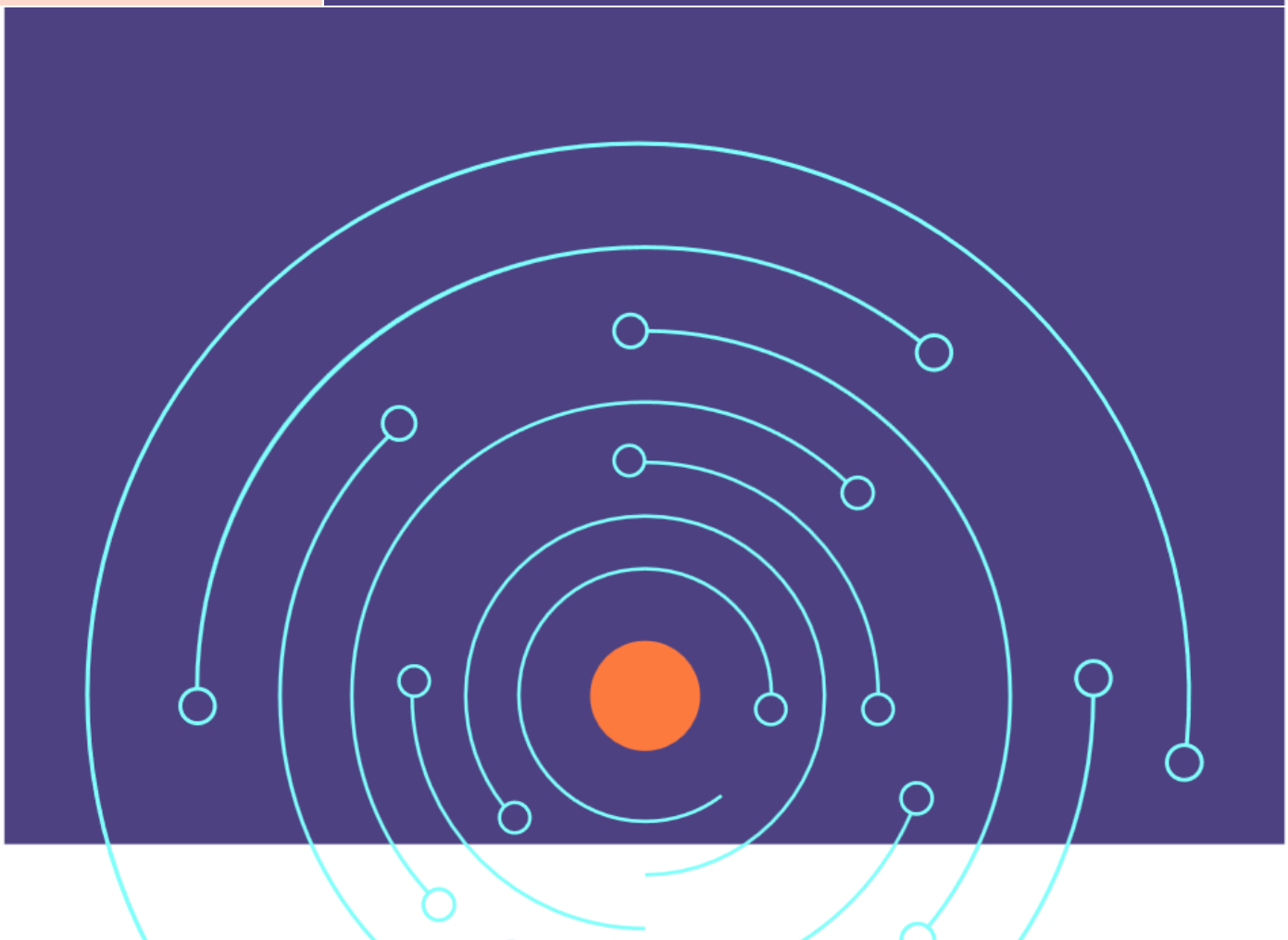

Table of contents

1. Component Description

This component utilizes a federated learning approach, which eliminates the need to centralize and process raw data from individual machines on an extremal server for training an ML model. Instead, multiple edge clients are equipped with isolated copies of the same ML model and perform dynamic retrains based on a pre-specified number of rounds at the factory site level. Clients distribute only model weights, and an aggregation event takes place at the server side during each training round, achieving increased security, abstraction, and a collaborative learning scheme.

CORE's methodology focuses on processing and analyzing data from multiple milling machines, leveraging federated learning to execute predictive maintenance tasks (e.g. estimating the remaining useful life, anomaly detection). This approach allows the final model to learn from data contributed by multiple machines, potentially enhancing the overall predictive capabilities and enabling more accurate maintenance strategies for minimizing downtime and optimizing milling operations.

2. Input

During each round of the model training stage, batches of **operational data** (provided by GF & FRAISA) from machines (e.g., vibration, torque, temperature, etc.) that execute their own milling process, **will be given as input to the local models** which have been deployed on the edge. Then, after local model training has been completed for each round, only the ML model parameters will be transmitted to the server's side for the aggregation step. This process will be repeated for a prespecified number of rounds every time the training phase has been scheduled to take place. After the aggregation step takes place on the server side the **updated global model weights return** to each edge device **as an input for the new training session.**

3. Output

After each local training cycle, the output comprises the essential Machine Learning model weights, which are transmitted to the server for the aggregation phase. Those weights, along with model performance metrics will be saved in a database located on the server and will be analyzed to identify patterns, trends, or anomalies in the training process. This analysis can help in understanding how the model is learning from different edge machines and if there are any specific insights to be gained regarding the federated learning process. Visualization tools can help make these patterns more understandable to stakeholders. After training and evaluation, the aggregated model will be deployed and made available for use. This model, which has learned from the data of multiple edge machines, can be used to facilitate informed decision-making by stakeholders in predictive maintenance tasks for each edge machine.

4. Information flow

1. Edge node collects operational data from the milling machine.
2. The data is pre-processed and then used as input for training the FL model.
3. Each node is trained locally on its collected data using its current model weights (which were sent originally after the global model was initialized on the central server).
4. All edge nodes send their respective model weight updates to the central server.
5. The server receives model updates from the selected client nodes and then aggregates these updates. This aggregation typically involves calculating the weighted average of the updates.
6. Once the global update has been computed the central server sends the updated global model weights back to each node.
7. Each edge node updates its local model with the new global weights received from the server.
8. The process above is repeated for a specified number of rounds.
9. After the training stage has been completed, model inference takes place until a trigger event initiates again the federated training process described above (e.g., data drift, new data available, etc.)
10. Model inference results are stored in a database and visualized.

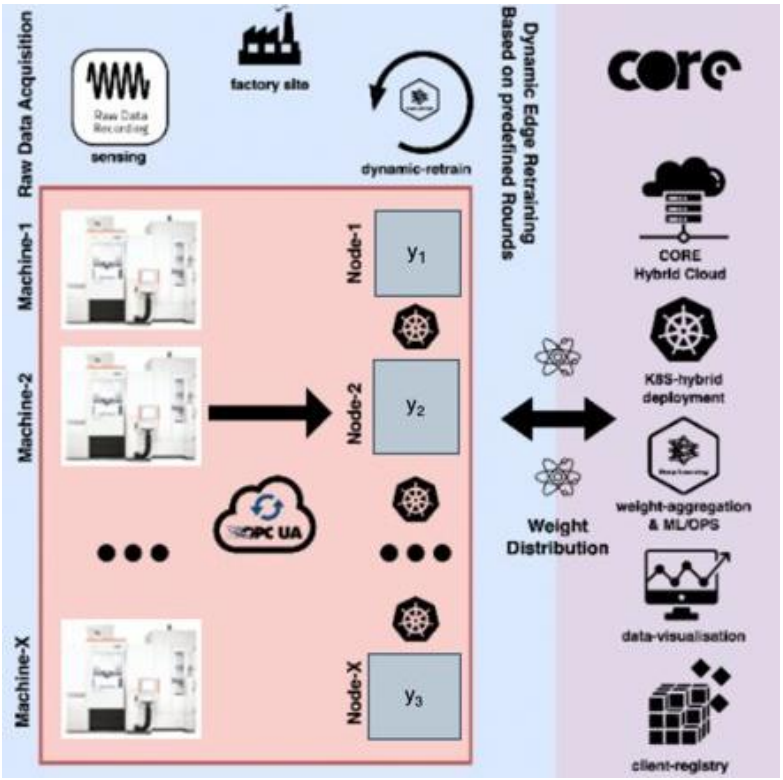

Figure 1 Federated Learning System Architecture

5. Internal Architecture

- Preprocessing Module: Cleans and structures data for input.
- Local Node Module: Processes data and computes local model updates.
- Server Module: Applies the aggregation step when receiving updates on local model parameters. This module is also responsible for sending the incoming inference result to the Storage Module.
- Communication Module: Sends model parameters to/from the central server from/to the edge nodes. It also sends the inference to the central server.
- Inference Module: Infers trained model using incoming data.
- Storage Module: Stores model inference output to a database

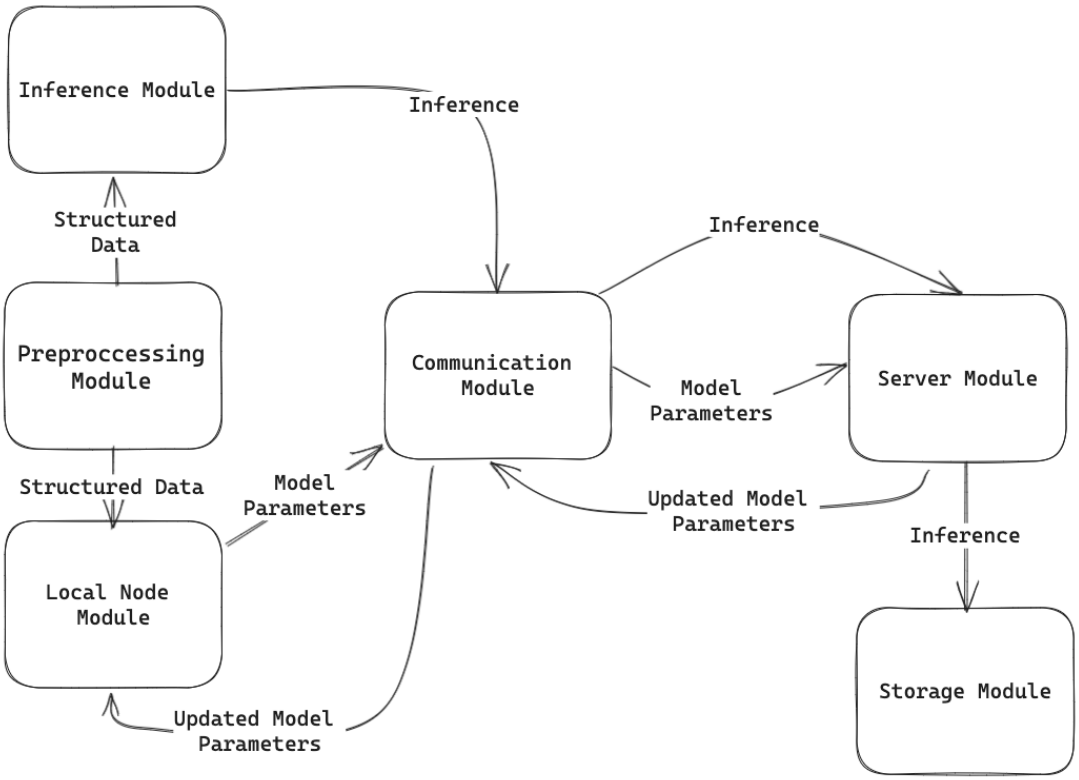

Figure 2 Interlinking of Internal Components: Communication Blueprint

6. API

- `preprocessData(rawData)`: Returns pre-processed data.
- `trainModel(data)`: Computes local model updates.
- `sendUpdates()`: Sends model updates to the central server.
- `receiveUpdates()`: Retrieves updated model weights from the server.
- `sendInference()`: Send output model inference.
- `saveInference()`: Store inference in database.

A more comprehensive API definition will be provided in the next stages of this component's development.

7. Implementation Technology

The selected programming language is Python, known for its readability and widespread use in data science and machine learning. The project also employs popular open-source libraries. TensorFlow or PyTorch are both powerful platforms for developing sophisticated Machine Learning models. The Flower framework plays a critical role in orchestrating Federated Learning processes, enabling the decentralized training of models across numerous devices, thereby enhancing data privacy, and optimizing communication costs.

A pivotal component in the setup is the message broker service, which could be either MQTT or Kafka, facilitating seamless and real-time transmission of model inference results to the central server. Triton Inference Server is an open-source inference serving software that streamlines and optimizes AI inferencing while managing requests in real-time. All logging details and model inference results are carefully stored and managed using InfluxDB, a high-performance time-series database.

8. Comments

The Flower framework selection speeds up the implementation of the Federated Learning architecture. Collaboration with GF and FRAISA is essential to ensure data quality and model effectiveness.