# A PROJECT REPORT
# ON

# DETECTION OF ABNORMAL WORKING CONDITION IN BOOST CONVERTER VIA DEEP LEARNING TECHNIQUES

BY
**ABHINAV ARYA**


**INDIAN INSTITUTE OF INFORMATION ROORKEE**
**UTTARAKHAND**

**JULY 2025**

# INTRODUCTION

Modern power electronics systems require not only efficiency but also high reliability and safety. One of the key components in such systems is the DC-DC Boost Converter, which steps up lower input voltages to higher output levels. These converters are widely used in electric vehicles, solar energy systems, embedded electronics, and robotics. However, like many real-world systems, they are vulnerable to faults and anomalies caused by sudden load changes, aging components, electrical noise, or environmental disturbances. Detecting these anomalies early is crucial to avoid potential damage or system failure.

This project addresses that need by developing a real-time anomaly detection system using machine learning. The solution integrates embedded signal acquisition, cloud-based model training, and edge deployment to monitor and detect abnormal behavior in DC-DC Boost Converter outputs. The core idea is to allow a deep learning model to learn from patterns in voltage signals and predict anomalies without relying on rigid thresholding or hand-crafted rules. The result is a system that is not only accurate and adaptive but also capable of running in real-time on low-power hardware.

The data acquisition is performed using an STM32 Nucleo-F303ZE microcontroller, which samples the output signal of the boost converter through an ADC pin. These analog readings are processed with basic filtering and transmitted over UART to a Raspberry Pi. The Raspberry Pi acts as the processing hub—receiving, interpreting, and classifying signal data using a deep learning model trained previously on labeled datasets. All incoming signals and predictions are logged automatically, enabling post-analysis and continuous improvement.

Model training is carried out on Kaggle using Python and PyTorch, where various architectures such as CNNs, ResNets, and MLPs are evaluated on real signal data collected during different operating conditions. Once the best-performing model is identified, it is converted into a deployable format and transferred to the Raspberry Pi for inference. The Raspberry Pi software includes tools to visualize the signal in real-time, run the model, and store logs for retraining. This closed-loop design enables seamless integration from data acquisition to real-time decision-making at the edge.

By combining embedded hardware with AI, this project demonstrates a practical and scalable approach to intelligent monitoring in power electronics. It is a valuable step toward smart diagnostics and predictive maintenance, especially in systems where traditional fault detection methods fall short. The design choices prioritize portability, cost-efficiency, and flexibility, making the system suitable for a wide range of applications in academia, industry, and research.

# LITERATURE REVIEW

Real-time anomaly detection in electrical systems has been an active area of research due to its potential to improve system reliability, efficiency, and safety. Traditional approaches to anomaly detection in DC-DC converters rely heavily on rule-based systems, mathematical modeling, or frequency domain analysis. These methods often require detailed system knowledge and are sensitive to noise or operating condition variations. For example, observer-based fault detection and threshold-based monitoring techniques have been widely used, but they tend to struggle in dynamic environments or under unknown fault conditions.

With the advent of machine learning, data-driven approaches have gained popularity in recent years. Techniques such as Principal Component Analysis (PCA), Support Vector Machines (SVM), and k-Nearest Neighbors (k-NN) have been applied to detect faults by learning patterns in historical data. However, these classical ML models often require significant feature engineering and are limited when dealing with non-linear, high-dimensional time series data common in power electronics. Additionally, their performance can degrade in real-time applications due to computational complexity or lack of temporal awareness.

Deep learning models, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have shown significant promise in handling time series signals and anomaly detection tasks. Studies such as Zhang et al. (2019) and Yadav et al. (2021) demonstrated the use of CNNs and LSTMs for real-time health monitoring in electrical machines and converters. These models can automatically extract features from raw signal data and generalize well across different operating conditions. U-Net architectures, originally developed for image segmentation, have also been adapted for 1D signal anomaly detection with promising results.

Edge AI and embedded deep learning have further extended the applicability of these models to low-power hardware platforms like Raspberry Pi and ARM Cortex-M microcontrollers. Recent advancements in frameworks like TensorFlow Lite and PyTorch Mobile have enabled the deployment of optimized neural networks on such platforms, making real-time anomaly detection feasible even in resource-constrained environments. This aligns with industry trends in Industry 4.0, where local intelligence at the sensor level is critical for predictive maintenance and fault prevention.

The proposed project builds upon these developments by combining STM32-based data acquisition, cloud-based model training, and Raspberry Pi-based inference. Unlike earlier works that often remain confined to simulation or offline testing, this system is deployed end-to-end in a real-time embedded environment. It leverages CNN-based deep learning models to detect anomalies directly from the time-domain voltage signal, demonstrating how AI-driven monitoring systems can be effectively implemented in real-world power electronic applications.

# MOTIVATION AND PROBLEM DEFINITION

Modern power electronic systems such as DC-DC converters are widely deployed in critical applications including electric vehicles, renewable energy systems, and industrial automation. As these systems become more complex and are expected to operate continuously under varying conditions, ensuring their reliability has become a key engineering challenge. Failures or anomalies in such systems can lead to reduced performance, increased maintenance costs, or even catastrophic damage. Traditional methods of anomaly detection rely heavily on rule-based systems or manual monitoring, which are not scalable or efficient for real-time, adaptive environments.

The need for intelligent, real-time monitoring solutions is growing rapidly, especially as industries move toward smart and autonomous systems under the umbrella of Industry 4.0. With the increasing availability of affordable microcontrollers and AI hardware, it has become technically feasible to implement edge intelligence—systems capable of detecting faults locally without needing cloud-based infrastructure. This opens the door for scalable, low-cost, and responsive anomaly detection solutions.

In this context, machine learning—and more specifically deep learning—offers a powerful alternative to traditional methods. Deep learning models can learn to recognize complex patterns and subtle deviations in raw signal data, without the need for handcrafted features or detailed system models. These models, once trained on labeled examples, can generalize to unseen scenarios and detect a wide variety of faults, including those caused by component degradation, load changes, or external disturbances. Deploying such models on edge devices like the Raspberry Pi allows for real-time inference close to the source of data, enabling quick response and decision-making.

The core problem this project addresses is how to accurately and efficiently detect anomalies in the output signal of a DC-DC Boost Converter in real time, using embedded hardware and deep learning. This involves several sub-challenges: acquiring clean signal data from an STM32 microcontroller, preprocessing and labeling data, training robust models on cloud platforms, and deploying those models to run reliably on lightweight hardware. The ultimate goal is to develop a

system that operates continuously and autonomously, identifying abnormal behavior without human intervention and logging relevant data for further analysis.

By addressing this problem, the project aims to demonstrate a complete and practical framework for embedded AI in power electronics—a framework that can be adapted or extended to other real-world systems requiring intelligent monitoring and fault detection.

# PROPOSED METHODOLOGY

The primary objective of this project is to develop a real-time anomaly detection system for DC-DC Boost Converter signals using deep learning and embedded hardware. The system integrates STM32 microcontroller for data acquisition, Raspberry Pi for edge computing and inference, and a 2D ResNet-based model trained to classify normal and anomalous signal patterns. The methodology is designed to be modular and replicable across similar applications involving time-series signal monitoring and fault detection.

The methodology begins with analog signal acquisition from the Boost Converter using the STM32 microcontroller. The signal, which can represent various operating states (e.g., normal operation, random perturbation, or load-induced fluctuations), is digitized through the ADC module and transmitted to the Raspberry Pi via UART communication at a baud rate of 115200.

On the Raspberry Pi, the serial data is logged into CSV files. These files are organized and labeled according to the type of signal behavior observed during collection. The data is then uploaded to Kaggle, where it is preprocessed—normalized, segmented into windows, and labeled—to train a deep learning model. The model chosen for this task is a 2D ResNet variant, adapted to handle reshaped one-dimensional time-series data as two-dimensional matrices.

Once trained, the model and the associated data scaler are exported and deployed back to the Raspberry Pi. The app.py script handles real-time UART data reception, preprocessing, inference using the deployed model, and prediction logging. This enables the system to continuously monitor the Boost Converter's behavior and detect anomalies in real-time, making it suitable for predictive maintenance or safety-critical applications.

## STM32 Microcontroller

The STM32 is a family of 32-bit microcontrollers developed by STMicroelectronics, widely used in industrial, automotive, and consumer electronics. For this project, the STM32 Nucleo-F303ZE board is used due to its powerful ARM Cortex-M4 processor with floating-point unit (FPU), high-resolution ADCs, and rich peripheral support. It serves as the sensing unit in the proposed system.

In the context of this project, the STM32 is programmed to read analog voltage levels from a DC-DC Boost Converter using its ADC1_IN5 channel (connected to pin PF4). The sampled data undergoes optional digital filtering in firmware (e.g., moving average) to remove high-frequency noise. The clean signal is then serialized and transmitted via UART using USART2 configured on pins PC4 (TX) and PC5 (RX).

The STM32 firmware is developed using STM32CubeIDE, which allows for easy configuration of clock sources, peripheral mappings, and interrupt settings. Using HAL (Hardware Abstraction Layer) libraries, UART and ADC peripherals are initialized with minimal code. The microcontroller continuously sends the ADC readings over UART at a fixed sampling rate, ensuring that data is streamed in real-time to the Raspberry Pi. This setup allows low-latency, low-power data acquisition suitable for embedded anomaly detection systems.

## Raspberry Pi

The Raspberry Pi is a compact and cost-effective Linux-based single-board computer that bridges the gap between embedded systems and full-fledged computing. In this project, it functions as the edge computing platform responsible for collecting serial data, preprocessing it, running inference using a trained deep learning model, and displaying or logging the results in real time.

The Raspberry Pi receives the serial data via its UART interface (TX: GPIO14, RX: GPIO15). A Python script running on the Pi (uart.py) continuously reads this data stream, timestamps each reading, and logs it into CSV format. These logs are essential for training and retraining machine learning models with real-world data.

For model inference, the Raspberry Pi runs a PyTorch-based script (inference.py) that loads a pre-trained model (model.pt) and a scaler object (scaler.pkl). As a batch of samples (typically 100 readings) is collected, the data is scaled and reshaped to match the model's input format, and inference is performed to detect anomalies. The result is displayed in the terminal and stored for audit or feedback.

Due to its support for Python, Torch, NumPy, and visualization libraries, the Raspberry Pi is ideal for deploying AI models at the edge. Its GPIO support also enables further extensions, such as triggering alarms or actuators based on the anomaly detection results.

## ResNet-2D Model

ResNet, or Residual Network, is a deep convolutional neural network architecture that addresses the vanishing gradient problem in deep networks using residual connections. For this project, a 2D version of ResNet is adapted to process time-series data by reshaping signal windows into pseudo-images, allowing convolutional layers to learn spatial patterns that represent temporal dynamics.

While traditional ResNet is designed for image classification, the project repurposes it by segmenting time-series signals into fixed-length windows (e.g., 100 samples) and converting them into 10x10 matrices or similar 2D shapes. This transformation allows the ResNet model to use its powerful convolutional filters to extract meaningful features from the signal patterns. Residual blocks in ResNet allow the network to retain low-level signal information even as deeper layers focus on higher-level abstractions.

The model is trained on labeled segments corresponding to different signal behaviors (e.g., normal, random, load disturbance) using cross-entropy loss and Adam optimizer in PyTorch. Once trained, the model is traced using TorchScript for efficient deployment on the Raspberry Pi. It demonstrates high accuracy in distinguishing between normal and anomalous signal patterns, even in noisy conditions.

The ResNet-2D model is a crucial component of the system, offering both robustness and scalability. Its architecture allows the model to generalize well to unseen signal patterns and to be retrained easily with updated field data.

# CONCLUSION AND FUTURE WORK

This project presents a complete pipeline for real-time anomaly detection in DC-DC Boost Converter signals using deep learning and embedded systems. The integration of STM32 for data acquisition, Raspberry Pi for edge inference, and a ResNet-2D model for classification creates a compact, efficient, and portable anomaly detection system. Through this setup, we demonstrated that it is possible to detect deviations in real-time voltage waveforms with high accuracy using trained deep learning models.

The system was validated through various signal scenarios including normal operation, sudden load changes, and random disturbances. With live UART communication between the STM32 and Raspberry Pi, and seamless data logging and prediction, the architecture is robust enough for real-world deployment in power electronics diagnostics, predictive maintenance, and educational setups.

Despite the system's strengths, there are several areas for further improvement. The current model is trained on limited datasets; larger and more diverse datasets will enhance model generalizability. Moreover, while ResNet-2D performs well, experimenting with other architectures like LSTM or Transformer-based models could better capture temporal dependencies in the signal.

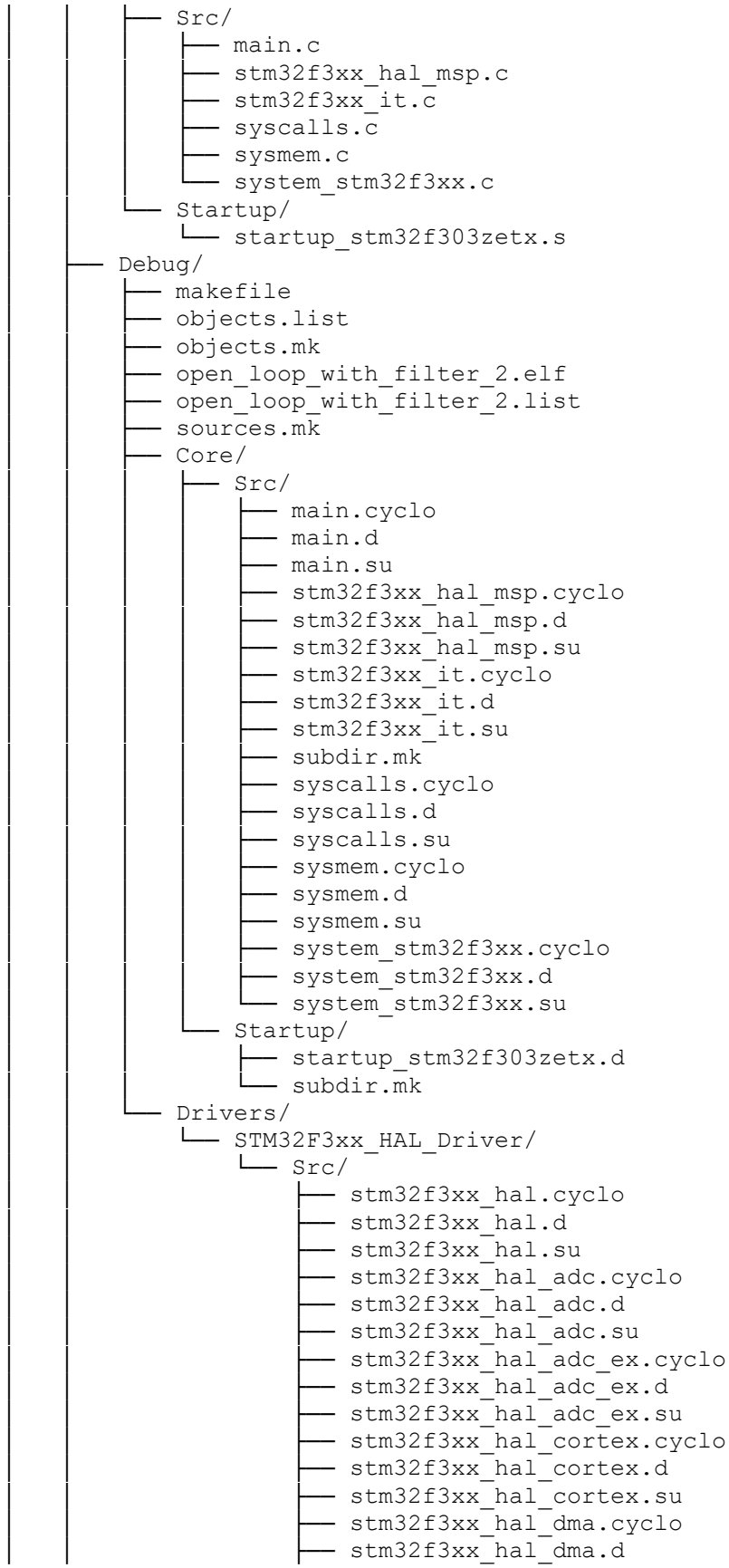
Future work could also focus on deploying the model on the STM32 itself using TinyML frameworks like TensorFlow Lite for Microcontrollers, making the entire system more compact and energy-efficient. Additionally, a web-based dashboard or mobile interface could be developed for remote monitoring and alerting. With such enhancements, this system can evolve into a fully autonomous, AI-powered diagnostic tool for smart grids and industrial power systems.

# REFERENCES


1. STMicroelectronics. *STM32 Nucleo-F303ZE Datasheet*. [Online]. Available: https://www.st.com/en/evaluation-tools/nucleo-f303ze.html
2. Raspberry Pi Foundation. *Raspberry Pi 4 Model B Documentation*. [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html
3. He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778.
4. PyTorch. *PyTorch Documentation*. [Online]. Available: https://pytorch.org/docs/stable/index.html
5. Kaggle. *Kaggle: Your Machine Learning and Data Science Community*. [Online]. Available: https://www.kaggle.com/
6. Serial Communication Protocols. *UART Communication Basics*. [Online]. Available: https://www.sparkfun.com/tutorials/65
7. TensorFlow Lite Micro. *TinyML on Microcontrollers*. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers
8. Scikit-learn Developers. *scikit-learn: Machine Learning in Python*. [Online]. Available: https://scikit-learn.org/

# CODE ATTACHMENTS

```
Directory structure:
└── abhinav3499-internship-iit-roorkee/
    ├── README.md
    ├── LICENSE
    ├── Dataset/
    │   ├── Boost Converter/
    │   │   ├── normal.csv
    │   │   └── random.csv
    │   └── DC Motor/
    │       ├── load.csv
    │       ├── normal.csv
    │       └── random.csv
    ├── Kaggle/
    │   └── model-training.ipynb
    ├── Raspberry Pi/
    │   ├── app.py
    │   ├── inference.py
    │   ├── plot.py
    │   ├── requirements.txt
    │   ├── uart.py
    │   ├── model_1/
    │   │   ├── resnet40_50.pt
    │   │   └── standard_scaler_40_50.pkl
    │   ├── model_2/
    │   │   ├── CNN_2D.pt
    │   │   └── standard_scaler_40_50.pkl
    │   ├── model_3/
    │   │   ├── standard_scaler_40_50.pkl
    │   │   └── VGG.pt
    │   ├── model_4/
    │   │   ├── standard_scaler_40_50.pkl
    │   │   └── UNet2D_40x50.pt
    │   ├── model_5/
    │   │   ├── MLP_2D.pt
    │   │   └── standard_scaler_40_50.pkl
    │   └── model_6/
    │       ├── model.pt
    │       ├── model_ResNet.pt
    │       ├── scaker.pkl
    │       └── scaler.pkl
    └── STM32/
        ├── Open Loop With Filter/
        │   ├── open_loop_with_filter_2 Debug.launch
        │   ├── open_loop_with_filter_2.ioc
        │   ├── STM32F303ZETX_FLASH.ld
        │   ├── .cproject
        │   ├── .mxproject
        │   ├── Core/
        │   │   ├── Inc/
        │   │   │   ├── main.h
        │   │   │   ├── stm32f3xx_hal_conf.h
        │   │   │   └── stm32f3xx_it.h
```

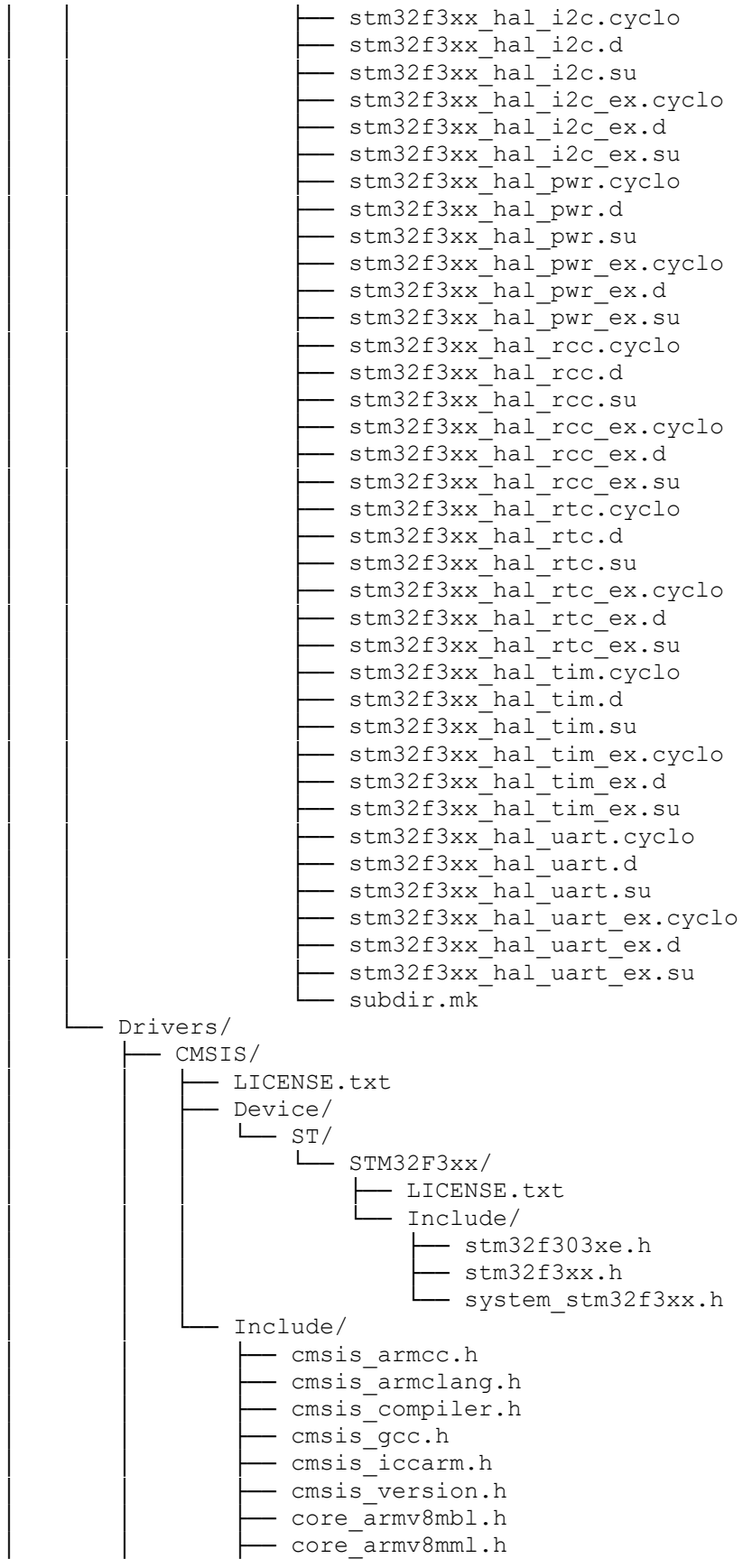

Src/
main.c
stm32f3xx_hal_msp.c
stm32f3xx_it.c
syscalls.c
sysmem.c
system_stm32f3xx.c
Startup/
startup_stm32f303zetx.s
Debug/
makefile
objects.list
objects.mk
open_loop_with_filter_2.elf
open_loop_with_filter_2.list
sources.mk
Core/
Src/
main.cyclo
main.d
main.su
stm32f3xx_hal_msp.cyclo
stm32f3xx_hal_msp.d
stm32f3xx_hal_msp.su
stm32f3xx_it.cyclo
stm32f3xx_it.d
stm32f3xx_it.su
subdir.mk
syscalls.cyclo
syscalls.d
syscalls.su
sysmem.cyclo
sysmem.d
sysmem.su
system_stm32f3xx.cyclo
system_stm32f3xx.d
system_stm32f3xx.su
Startup/
startup_stm32f303zetx.d
subdir.mk
Drivers/
STM32F3xx_HAL_Driver/
Src/
stm32f3xx_hal.cyclo
stm32f3xx_hal.d
stm32f3xx_hal.su
stm32f3xx_hal_adc.cyclo
stm32f3xx_hal_adc.d
stm32f3xx_hal_adc.su
stm32f3xx_hal_adc_ex.cyclo
stm32f3xx_hal_adc_ex.d
stm32f3xx_hal_adc_ex.su
stm32f3xx_hal_cortex.cyclo
stm32f3xx_hal_cortex.d
stm32f3xx_hal_cortex.su
stm32f3xx_hal_dma.cyclo
stm32f3xx_hal_dma.d

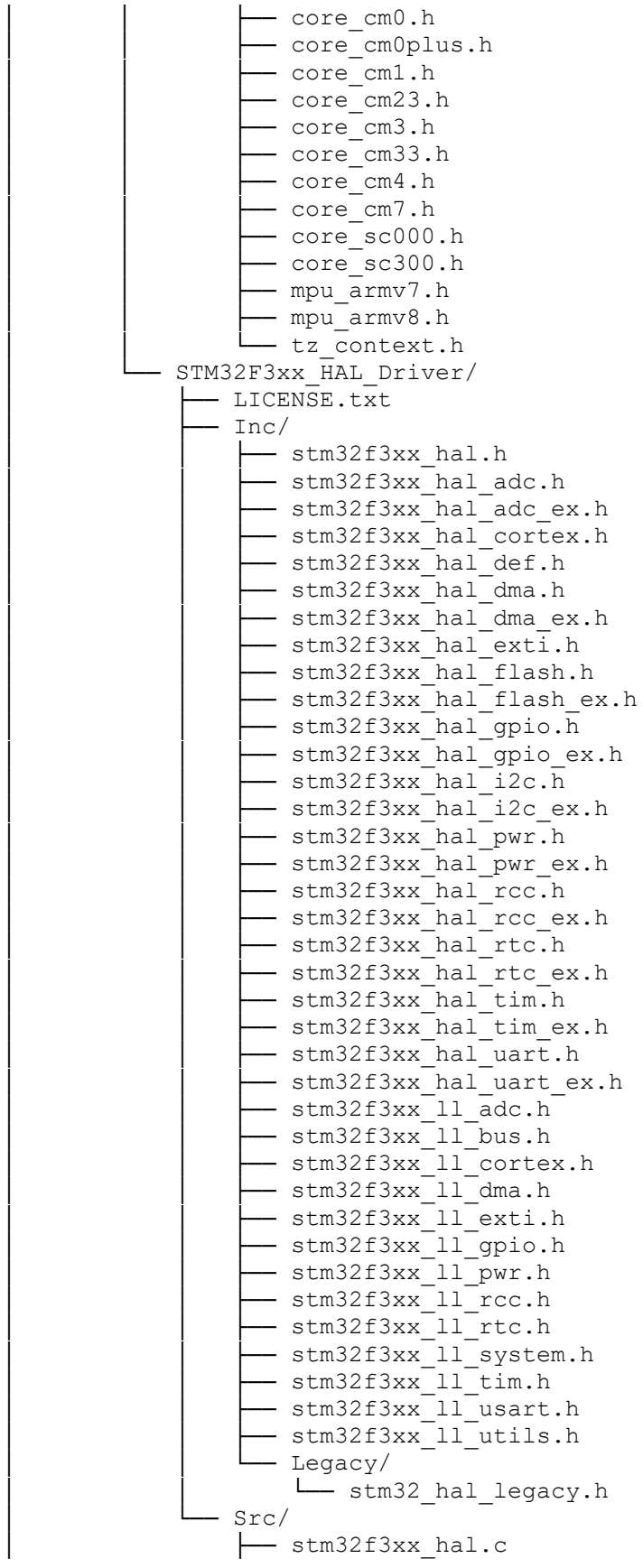
```
│   │   ├── stm32f3xx_hal_dma.su
│   │   ├── stm32f3xx_hal_exti.cyclo
│   │   ├── stm32f3xx_hal_exti.d
│   │   ├── stm32f3xx_hal_exti.su
│   │   ├── stm32f3xx_hal_flash.cyclo
│   │   ├── stm32f3xx_hal_flash.d
│   │   ├── stm32f3xx_hal_flash.su
│   │   ├── stm32f3xx_hal_flash_ex.cyclo
│   │   ├── stm32f3xx_hal_flash_ex.d
│   │   ├── stm32f3xx_hal_flash_ex.su
│   │   ├── stm32f3xx_hal_gpio.cyclo
│   │   ├── stm32f3xx_hal_gpio.d
│   │   ├── stm32f3xx_hal_gpio.su
│   │   ├── stm32f3xx_hal_i2c.cyclo
│   │   ├── stm32f3xx_hal_i2c.d
│   │   ├── stm32f3xx_hal_i2c.su
│   │   ├── stm32f3xx_hal_i2c_ex.cyclo
│   │   ├── stm32f3xx_hal_i2c_ex.d
│   │   ├── stm32f3xx_hal_i2c_ex.su
│   │   ├── stm32f3xx_hal_pwr.cyclo
│   │   ├── stm32f3xx_hal_pwr.d
│   │   ├── stm32f3xx_hal_pwr.su
│   │   ├── stm32f3xx_hal_pwr_ex.cyclo
│   │   ├── stm32f3xx_hal_pwr_ex.d
│   │   ├── stm32f3xx_hal_pwr_ex.su
│   │   ├── stm32f3xx_hal_rcc.cyclo
│   │   ├── stm32f3xx_hal_rcc.d
│   │   ├── stm32f3xx_hal_rcc.su
│   │   ├── stm32f3xx_hal_rcc_ex.cyclo
│   │   ├── stm32f3xx_hal_rcc_ex.d
│   │   ├── stm32f3xx_hal_rcc_ex.su
│   │   ├── stm32f3xx_hal_rtc.cyclo
│   │   ├── stm32f3xx_hal_rtc.d
│   │   ├── stm32f3xx_hal_rtc.su
│   │   ├── stm32f3xx_hal_rtc_ex.cyclo
│   │   ├── stm32f3xx_hal_rtc_ex.d
│   │   ├── stm32f3xx_hal_rtc_ex.su
│   │   ├── stm32f3xx_hal_tim.cyclo
│   │   ├── stm32f3xx_hal_tim.d
│   │   ├── stm32f3xx_hal_tim.su
│   │   ├── stm32f3xx_hal_tim_ex.cyclo
│   │   ├── stm32f3xx_hal_tim_ex.d
│   │   ├── stm32f3xx_hal_tim_ex.su
│   │   ├── stm32f3xx_hal_uart.cyclo
│   │   ├── stm32f3xx_hal_uart.d
│   │   ├── stm32f3xx_hal_uart.su
│   │   ├── stm32f3xx_hal_uart_ex.cyclo
│   │   ├── stm32f3xx_hal_uart_ex.d
│   │   ├── stm32f3xx_hal_uart_ex.su
│   │   └── subdir.mk
│   └── Drivers/
│       ├── CMSIS/
│       │   ├── LICENSE.txt
│       │   ├── Device/
│       │   │   └── ST/
│       │   │       └── STM32F3xx/
│       │   │           ├── LICENSE.txt
```

```
│   │   │   │   └── Include/
│   │   │   │       ├── stm32f303xe.h
│   │   │   │       ├── stm32f3xx.h
│   │   │   │       └── system_stm32f3xx.h
│   │   │   └── Include/
│   │   │       ├── cmsis_armcc.h
│   │   │       ├── cmsis_armclang.h
│   │   │       ├── cmsis_compiler.h
│   │   │       ├── cmsis_gcc.h
│   │   │       ├── cmsis_iccarm.h
│   │   │       ├── cmsis_version.h
│   │   │       ├── core_armv8mbl.h
│   │   │       ├── core_armv8mml.h
│   │   │       ├── core_cm0.h
│   │   │       ├── core_cm0plus.h
│   │   │       ├── core_cm1.h
│   │   │       ├── core_cm23.h
│   │   │       ├── core_cm3.h
│   │   │       ├── core_cm33.h
│   │   │       ├── core_cm4.h
│   │   │       ├── core_cm7.h
│   │   │       ├── core_sc000.h
│   │   │       ├── core_sc300.h
│   │   │       ├── mpu_armv7.h
│   │   │       ├── mpu_armv8.h
│   │   │       └── tz_context.h
│   │   └── STM32F3xx_HAL_Driver/
│   │       ├── LICENSE.txt
│   │       ├── Inc/
│   │       │   ├── stm32f3xx_hal.h
│   │       │   ├── stm32f3xx_hal_adc.h
│   │       │   ├── stm32f3xx_hal_adc_ex.h
│   │       │   ├── stm32f3xx_hal_cortex.h
│   │       │   ├── stm32f3xx_hal_def.h
│   │       │   ├── stm32f3xx_hal_dma.h
│   │       │   ├── stm32f3xx_hal_dma_ex.h
│   │       │   ├── stm32f3xx_hal_exti.h
│   │       │   ├── stm32f3xx_hal_flash.h
│   │       │   ├── stm32f3xx_hal_flash_ex.h
│   │       │   ├── stm32f3xx_hal_gpio.h
│   │       │   ├── stm32f3xx_hal_gpio_ex.h
│   │       │   ├── stm32f3xx_hal_i2c.h
│   │       │   ├── stm32f3xx_hal_i2c_ex.h
│   │       │   ├── stm32f3xx_hal_pwr.h
│   │       │   ├── stm32f3xx_hal_pwr_ex.h
│   │       │   ├── stm32f3xx_hal_rcc.h
│   │       │   ├── stm32f3xx_hal_rcc_ex.h
│   │       │   ├── stm32f3xx_hal_rtc.h
│   │       │   ├── stm32f3xx_hal_rtc_ex.h
│   │       │   ├── stm32f3xx_hal_tim.h
│   │       │   ├── stm32f3xx_hal_tim_ex.h
│   │       │   ├── stm32f3xx_hal_uart.h
│   │       │   ├── stm32f3xx_hal_uart_ex.h
│   │       │   ├── stm32f3xx_ll_adc.h
│   │       │   ├── stm32f3xx_ll_bus.h
│   │       │   ├── stm32f3xx_ll_cortex.h
│   │       │   ├── stm32f3xx_ll_dma.h
```

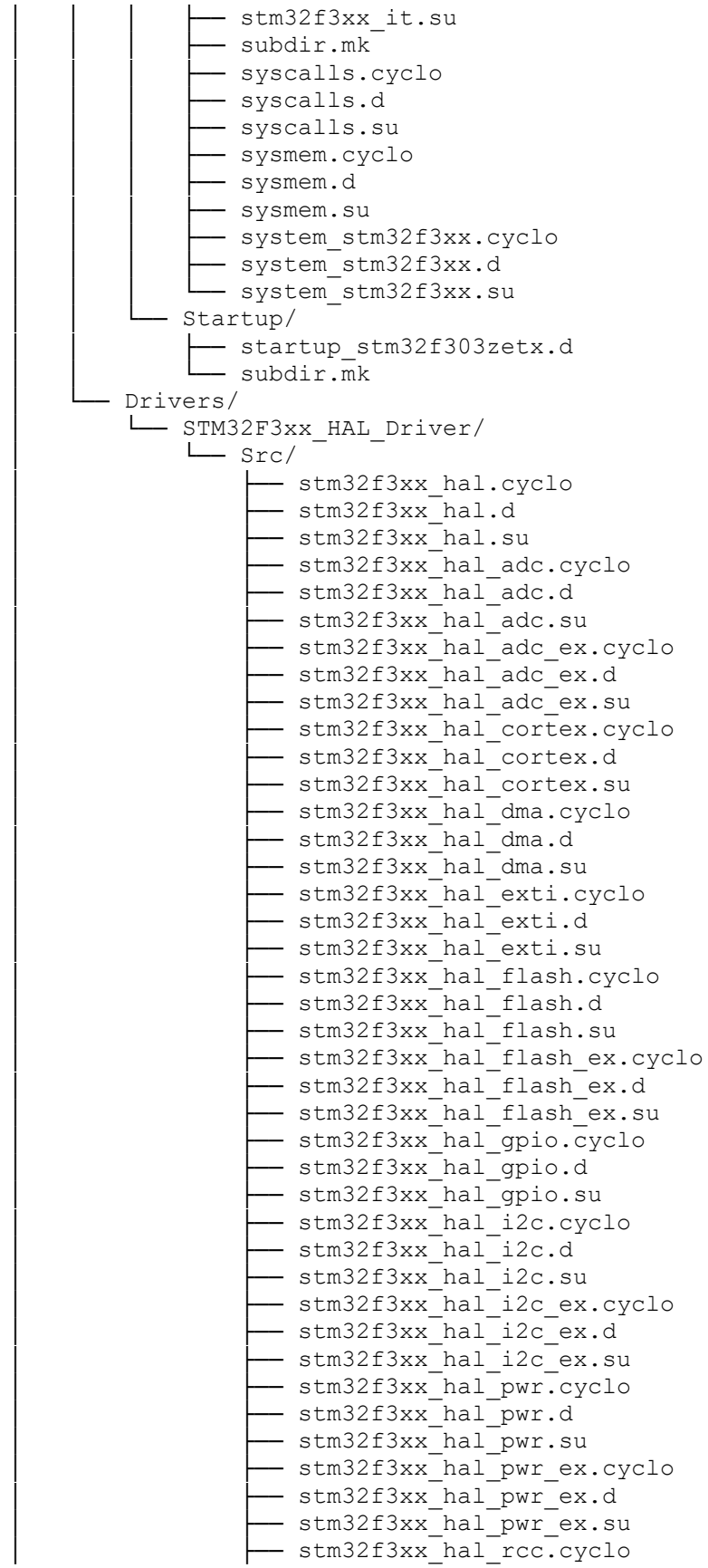
```
│   │   │   ├── stm32f3xx_ll_exti.h
│   │   │   ├── stm32f3xx_ll_gpio.h
│   │   │   ├── stm32f3xx_ll_pwr.h
│   │   │   ├── stm32f3xx_ll_rcc.h
│   │   │   ├── stm32f3xx_ll_rtc.h
│   │   │   ├── stm32f3xx_ll_system.h
│   │   │   ├── stm32f3xx_ll_tim.h
│   │   │   ├── stm32f3xx_ll_usart.h
│   │   │   ├── stm32f3xx_ll_utils.h
│   │   │   └── Legacy/
│   │   │       └── stm32_hal_legacy.h
│   │   └── Src/
│   │       ├── stm32f3xx_hal.c
│   │       ├── stm32f3xx_hal_adc.c
│   │       ├── stm32f3xx_hal_adc_ex.c
│   │       ├── stm32f3xx_hal_cortex.c
│   │       ├── stm32f3xx_hal_dma.c
│   │       ├── stm32f3xx_hal_exti.c
│   │       ├── stm32f3xx_hal_flash.c
│   │       ├── stm32f3xx_hal_flash_ex.c
│   │       ├── stm32f3xx_hal_gpio.c
│   │       ├── stm32f3xx_hal_i2c.c
│   │       ├── stm32f3xx_hal_i2c_ex.c
│   │       ├── stm32f3xx_hal_pwr.c
│   │       ├── stm32f3xx_hal_pwr_ex.c
│   │       ├── stm32f3xx_hal_rcc.c
│   │       ├── stm32f3xx_hal_rcc_ex.c
│   │       ├── stm32f3xx_hal_rtc.c
│   │       ├── stm32f3xx_hal_rtc_ex.c
│   │       ├── stm32f3xx_hal_tim.c
│   │       ├── stm32f3xx_hal_tim_ex.c
│   │       ├── stm32f3xx_hal_uart.c
│   │       └── stm32f3xx_hal_uart_ex.c
├── Open Loop Without Filter/
│   ├── open_loop_without_filter Debug.launch
│   ├── open_loop_without_filter.ioc
│   ├── STM32F303ZETX_FLASH.ld
│   ├── .cproject
│   ├── .mxproject
│   ├── Core/
│   │   ├── Inc/
│   │   │   ├── main.h
│   │   │   ├── stm32f3xx_hal_conf.h
│   │   │   └── stm32f3xx_it.h
│   │   ├── Src/
│   │   │   ├── main.c
│   │   │   ├── stm32f3xx_hal_msp.c
│   │   │   ├── stm32f3xx_it.c
│   │   │   ├── syscalls.c
│   │   │   ├── sysmem.c
│   │   │   └── system_stm32f3xx.c
│   │   └── Startup/
│   │       └── startup_stm32f303zetx.s
│   ├── Debug/
│   │   ├── makefile
│   │   ├── objects.list
│   │   ├── objects.mk
```

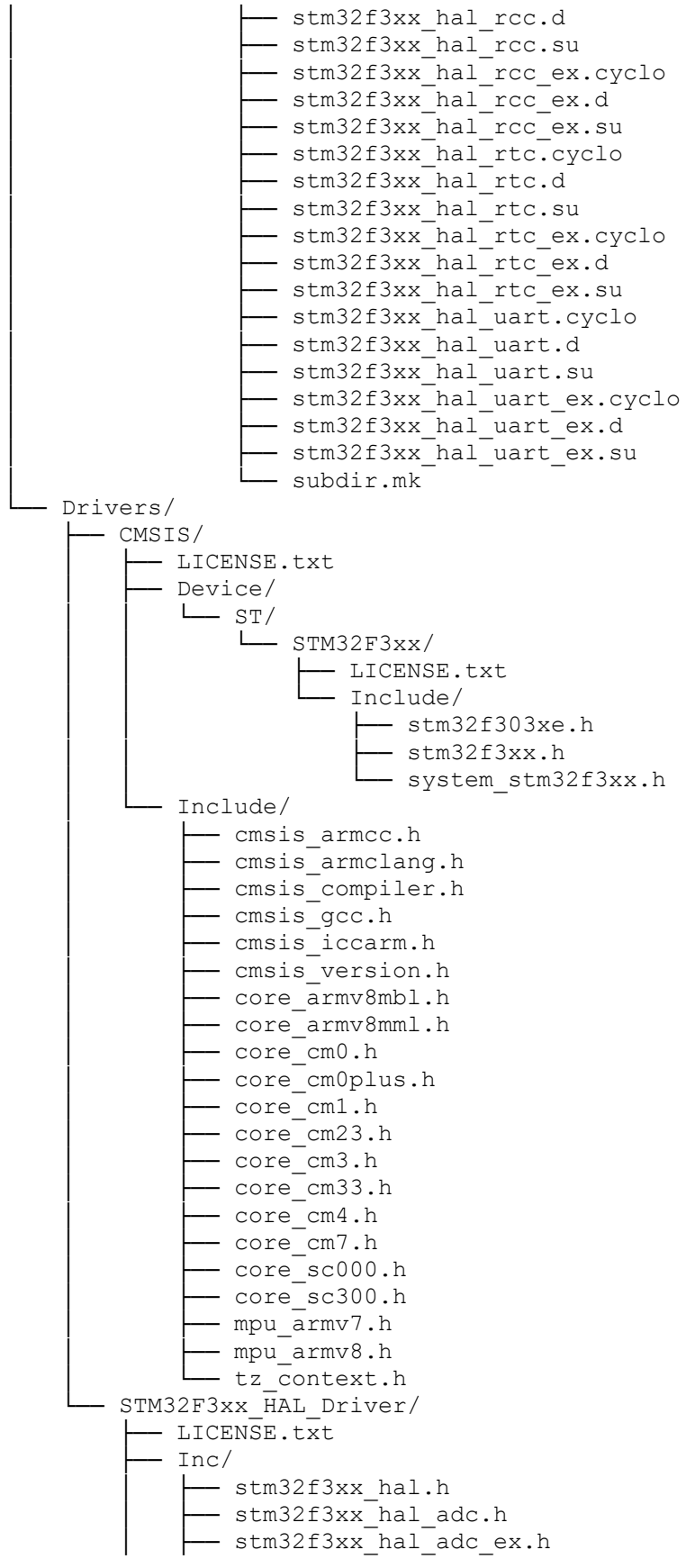
```
|   |   ├── open_loop_without_filter.elf
|   |   ├── open_loop_without_filter.list
|   |   ├── sources.mk
|   |   ├── Core/
|   |   |   ├── Src/
|   |   |   |   ├── main.cyclo
|   |   |   |   ├── main.d
|   |   |   |   ├── main.su
|   |   |   |   ├── stm32f3xx_hal_msp.cyclo
|   |   |   |   ├── stm32f3xx_hal_msp.d
|   |   |   |   ├── stm32f3xx_hal_msp.su
|   |   |   |   ├── stm32f3xx_it.cyclo
|   |   |   |   ├── stm32f3xx_it.d
|   |   |   |   ├── stm32f3xx_it.su
|   |   |   |   ├── subdir.mk
|   |   |   |   ├── syscalls.cyclo
|   |   |   |   ├── syscalls.d
|   |   |   |   ├── syscalls.su
|   |   |   |   ├── sysmem.cyclo
|   |   |   |   ├── sysmem.d
|   |   |   |   ├── sysmem.su
|   |   |   |   ├── system_stm32f3xx.cyclo
|   |   |   |   ├── system_stm32f3xx.d
|   |   |   |   └── system_stm32f3xx.su
|   |   |   └── Startup/
|   |   |       ├── startup_stm32f303zetx.d
|   |   |       └── subdir.mk
|   |   └── Drivers/
|   |       └── STM32F3xx_HAL_Driver/
|   |           └── Src/
|   |               ├── stm32f3xx_hal.cyclo
|   |               ├── stm32f3xx_hal.d
|   |               ├── stm32f3xx_hal.su
|   |               ├── stm32f3xx_hal_adc.cyclo
|   |               ├── stm32f3xx_hal_adc.d
|   |               ├── stm32f3xx_hal_adc.su
|   |               ├── stm32f3xx_hal_adc_ex.cyclo
|   |               ├── stm32f3xx_hal_adc_ex.d
|   |               ├── stm32f3xx_hal_adc_ex.su
|   |               ├── stm32f3xx_hal_cortex.cyclo
|   |               ├── stm32f3xx_hal_cortex.d
|   |               ├── stm32f3xx_hal_cortex.su
|   |               ├── stm32f3xx_hal_dma.cyclo
|   |               ├── stm32f3xx_hal_dma.d
|   |               ├── stm32f3xx_hal_dma.su
|   |               ├── stm32f3xx_hal_exti.cyclo
|   |               ├── stm32f3xx_hal_exti.d
|   |               ├── stm32f3xx_hal_exti.su
|   |               ├── stm32f3xx_hal_flash.cyclo
|   |               ├── stm32f3xx_hal_flash.d
|   |               ├── stm32f3xx_hal_flash.su
|   |               ├── stm32f3xx_hal_flash_ex.cyclo
|   |               ├── stm32f3xx_hal_flash_ex.d
|   |               ├── stm32f3xx_hal_flash_ex.su
|   |               ├── stm32f3xx_hal_gpio.cyclo
|   |               ├── stm32f3xx_hal_gpio.d
|   |               ├── stm32f3xx_hal_gpio.su
```

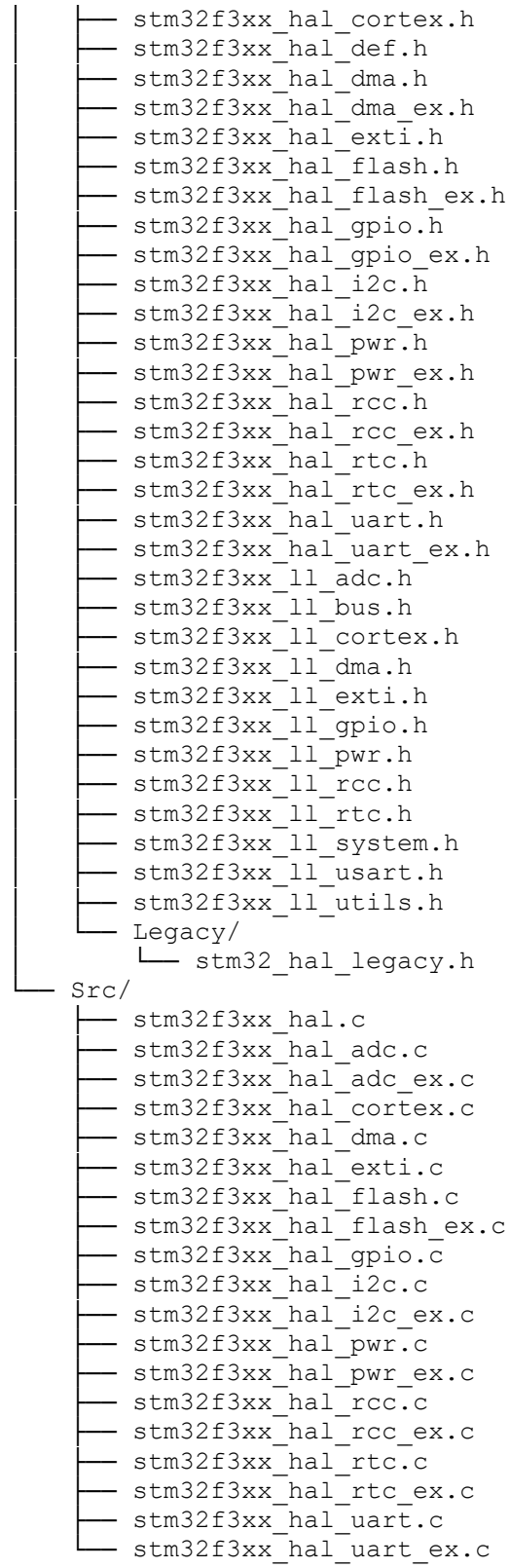

stm32f3xx_hal_i2c.cyclo
stm32f3xx_hal_i2c.d
stm32f3xx_hal_i2c.su
stm32f3xx_hal_i2c_ex.cyclo
stm32f3xx_hal_i2c_ex.d
stm32f3xx_hal_i2c_ex.su
stm32f3xx_hal_pwr.cyclo
stm32f3xx_hal_pwr.d
stm32f3xx_hal_pwr.su
stm32f3xx_hal_pwr_ex.cyclo
stm32f3xx_hal_pwr_ex.d
stm32f3xx_hal_pwr_ex.su
stm32f3xx_hal_rcc.cyclo
stm32f3xx_hal_rcc.d
stm32f3xx_hal_rcc.su
stm32f3xx_hal_rcc_ex.cyclo
stm32f3xx_hal_rcc_ex.d
stm32f3xx_hal_rcc_ex.su
stm32f3xx_hal_rtc.cyclo
stm32f3xx_hal_rtc.d
stm32f3xx_hal_rtc.su
stm32f3xx_hal_rtc_ex.cyclo
stm32f3xx_hal_rtc_ex.d
stm32f3xx_hal_rtc_ex.su
stm32f3xx_hal_tim.cyclo
stm32f3xx_hal_tim.d
stm32f3xx_hal_tim.su
stm32f3xx_hal_tim_ex.cyclo
stm32f3xx_hal_tim_ex.d
stm32f3xx_hal_tim_ex.su
stm32f3xx_hal_uart.cyclo
stm32f3xx_hal_uart.d
stm32f3xx_hal_uart.su
stm32f3xx_hal_uart_ex.cyclo
stm32f3xx_hal_uart_ex.d
stm32f3xx_hal_uart_ex.su
subdir.mk
Drivers/
CMSIS/
LICENSE.txt
Device/
ST/
STM32F3xx/
LICENSE.txt
Include/
stm32f303xe.h
stm32f3xx.h
system_stm32f3xx.h
Include/
cmsis_armcc.h
cmsis_armclang.h
cmsis_compiler.h
cmsis_gcc.h
cmsis_iccarm.h
cmsis_version.h
core_armv8mbl.h
core_armv8mml.h


core_cm0.h
core_cm0plus.h
core_cm1.h
core_cm23.h
core_cm3.h
core_cm33.h
core_cm4.h
core_cm7.h
core_sc000.h
core_sc300.h
mpu_armv7.h
mpu_armv8.h
tz_context.h
STM32F3xx_HAL_Driver/
LICENSE.txt
Inc/
stm32f3xx_hal.h
stm32f3xx_hal_adc.h
stm32f3xx_hal_adc_ex.h
stm32f3xx_hal_cortex.h
stm32f3xx_hal_def.h
stm32f3xx_hal_dma.h
stm32f3xx_hal_dma_ex.h
stm32f3xx_hal_exti.h
stm32f3xx_hal_flash.h
stm32f3xx_hal_flash_ex.h
stm32f3xx_hal_gpio.h
stm32f3xx_hal_gpio_ex.h
stm32f3xx_hal_i2c.h
stm32f3xx_hal_i2c_ex.h
stm32f3xx_hal_pwr.h
stm32f3xx_hal_pwr_ex.h
stm32f3xx_hal_rcc.h
stm32f3xx_hal_rcc_ex.h
stm32f3xx_hal_rtc.h
stm32f3xx_hal_rtc_ex.h
stm32f3xx_hal_tim.h
stm32f3xx_hal_tim_ex.h
stm32f3xx_hal_uart.h
stm32f3xx_hal_uart_ex.h
stm32f3xx_ll_adc.h
stm32f3xx_ll_bus.h
stm32f3xx_ll_cortex.h
stm32f3xx_ll_dma.h
stm32f3xx_ll_exti.h
stm32f3xx_ll_gpio.h
stm32f3xx_ll_pwr.h
stm32f3xx_ll_rcc.h
stm32f3xx_ll_rtc.h
stm32f3xx_ll_system.h
stm32f3xx_ll_tim.h
stm32f3xx_ll_usart.h
stm32f3xx_ll_utils.h
Legacy/
stm32_hal_legacy.h
Src/
stm32f3xx_hal.c

```
│               ├── stm32f3xx_hal_adc.c
│               ├── stm32f3xx_hal_adc_ex.c
│               ├── stm32f3xx_hal_cortex.c
│               ├── stm32f3xx_hal_dma.c
│               ├── stm32f3xx_hal_exti.c
│               ├── stm32f3xx_hal_flash.c
│               ├── stm32f3xx_hal_flash_ex.c
│               ├── stm32f3xx_hal_gpio.c
│               ├── stm32f3xx_hal_i2c.c
│               ├── stm32f3xx_hal_i2c_ex.c
│               ├── stm32f3xx_hal_pwr.c
│               ├── stm32f3xx_hal_pwr_ex.c
│               ├── stm32f3xx_hal_rcc.c
│               ├── stm32f3xx_hal_rcc_ex.c
│               ├── stm32f3xx_hal_rtc.c
│               ├── stm32f3xx_hal_rtc_ex.c
│               ├── stm32f3xx_hal_tim.c
│               ├── stm32f3xx_hal_tim_ex.c
│               ├── stm32f3xx_hal_uart.c
│               └── stm32f3xx_hal_uart_ex.c
└── UART Data Reading/
    ├── STM32F303ZETX_FLASH.ld
    ├── uart2 Debug.launch
    ├── uart2.ioc
    ├── .cproject
    ├── .mxproject
    ├── Core/
    │   ├── Inc/
    │   │   ├── main.h
    │   │   ├── stm32f3xx_hal_conf.h
    │   │   └── stm32f3xx_it.h
    │   ├── Src/
    │   │   ├── main.c
    │   │   ├── stm32f3xx_hal_msp.c
    │   │   ├── stm32f3xx_it.c
    │   │   ├── syscalls.c
    │   │   ├── sysmem.c
    │   │   └── system_stm32f3xx.c
    │   └── Startup/
    │       └── startup_stm32f303zetx.s
    ├── Debug/
    │   ├── makefile
    │   ├── objects.list
    │   ├── objects.mk
    │   ├── sources.mk
    │   ├── uart2.elf
    │   ├── uart2.list
    │   ├── Core/
    │   │   ├── Src/
    │   │   │   ├── main.cyclo
    │   │   │   ├── main.d
    │   │   │   ├── main.su
    │   │   │   ├── stm32f3xx_hal_msp.cyclo
    │   │   │   ├── stm32f3xx_hal_msp.d
    │   │   │   ├── stm32f3xx_hal_msp.su
    │   │   │   ├── stm32f3xx_it.cyclo
    │   │   │   ├── stm32f3xx_it.d
```


stm32f3xx_it.su
subdir.mk
syscalls.cyclo
syscalls.d
syscalls.su
sysmem.cyclo
sysmem.d
sysmem.su
system_stm32f3xx.cyclo
system_stm32f3xx.d
system_stm32f3xx.su
Startup/
startup_stm32f303zetx.d
subdir.mk
Drivers/
STM32F3xx_HAL_Driver/
Src/
stm32f3xx_hal.cyclo
stm32f3xx_hal.d
stm32f3xx_hal.su
stm32f3xx_hal_adc.cyclo
stm32f3xx_hal_adc.d
stm32f3xx_hal_adc.su
stm32f3xx_hal_adc_ex.cyclo
stm32f3xx_hal_adc_ex.d
stm32f3xx_hal_adc_ex.su
stm32f3xx_hal_cortex.cyclo
stm32f3xx_hal_cortex.d
stm32f3xx_hal_cortex.su
stm32f3xx_hal_dma.cyclo
stm32f3xx_hal_dma.d
stm32f3xx_hal_dma.su
stm32f3xx_hal_exti.cyclo
stm32f3xx_hal_exti.d
stm32f3xx_hal_exti.su
stm32f3xx_hal_flash.cyclo
stm32f3xx_hal_flash.d
stm32f3xx_hal_flash.su
stm32f3xx_hal_flash_ex.cyclo
stm32f3xx_hal_flash_ex.d
stm32f3xx_hal_flash_ex.su
stm32f3xx_hal_gpio.cyclo
stm32f3xx_hal_gpio.d
stm32f3xx_hal_gpio.su
stm32f3xx_hal_i2c.cyclo
stm32f3xx_hal_i2c.d
stm32f3xx_hal_i2c.su
stm32f3xx_hal_i2c_ex.cyclo
stm32f3xx_hal_i2c_ex.d
stm32f3xx_hal_i2c_ex.su
stm32f3xx_hal_pwr.cyclo
stm32f3xx_hal_pwr.d
stm32f3xx_hal_pwr.su
stm32f3xx_hal_pwr_ex.cyclo
stm32f3xx_hal_pwr_ex.d
stm32f3xx_hal_pwr_ex.su
stm32f3xx_hal_rcc.cyclo


stm32f3xx_hal_rcc.d
stm32f3xx_hal_rcc.su
stm32f3xx_hal_rcc_ex.cyclo
stm32f3xx_hal_rcc_ex.d
stm32f3xx_hal_rcc_ex.su
stm32f3xx_hal_rtc.cyclo
stm32f3xx_hal_rtc.d
stm32f3xx_hal_rtc.su
stm32f3xx_hal_rtc_ex.cyclo
stm32f3xx_hal_rtc_ex.d
stm32f3xx_hal_rtc_ex.su
stm32f3xx_hal_uart.cyclo
stm32f3xx_hal_uart.d
stm32f3xx_hal_uart.su
stm32f3xx_hal_uart_ex.cyclo
stm32f3xx_hal_uart_ex.d
stm32f3xx_hal_uart_ex.su
subdir.mk
Drivers/
CMSIS/
LICENSE.txt
Device/
ST/
STM32F3xx/
LICENSE.txt
Include/
stm32f303xe.h
stm32f3xx.h
system_stm32f3xx.h
Include/
cmsis_armcc.h
cmsis_armclang.h
cmsis_compiler.h
cmsis_gcc.h
cmsis_iccarm.h
cmsis_version.h
core_armv8mbl.h
core_armv8mml.h
core_cm0.h
core_cm0plus.h
core_cm1.h
core_cm23.h
core_cm3.h
core_cm33.h
core_cm4.h
core_cm7.h
core_sc000.h
core_sc300.h
mpu_armv7.h
mpu_armv8.h
tz_context.h
STM32F3xx_HAL_Driver/
LICENSE.txt
Inc/
stm32f3xx_hal.h
stm32f3xx_hal_adc.h
stm32f3xx_hal_adc_ex.h


stm32f3xx_hal_cortex.h
stm32f3xx_hal_def.h
stm32f3xx_hal_dma.h
stm32f3xx_hal_dma_ex.h
stm32f3xx_hal_exti.h
stm32f3xx_hal_flash.h
stm32f3xx_hal_flash_ex.h
stm32f3xx_hal_gpio.h
stm32f3xx_hal_gpio_ex.h
stm32f3xx_hal_i2c.h
stm32f3xx_hal_i2c_ex.h
stm32f3xx_hal_pwr.h
stm32f3xx_hal_pwr_ex.h
stm32f3xx_hal_rcc.h
stm32f3xx_hal_rcc_ex.h
stm32f3xx_hal_rtc.h
stm32f3xx_hal_rtc_ex.h
stm32f3xx_hal_uart.h
stm32f3xx_hal_uart_ex.h
stm32f3xx_ll_adc.h
stm32f3xx_ll_bus.h
stm32f3xx_ll_cortex.h
stm32f3xx_ll_dma.h
stm32f3xx_ll_exti.h
stm32f3xx_ll_gpio.h
stm32f3xx_ll_pwr.h
stm32f3xx_ll_rcc.h
stm32f3xx_ll_rtc.h
stm32f3xx_ll_system.h
stm32f3xx_ll_usart.h
stm32f3xx_ll_utils.h
Legacy/
stm32_hal_legacy.h
Src/
stm32f3xx_hal.c
stm32f3xx_hal_adc.c
stm32f3xx_hal_adc_ex.c
stm32f3xx_hal_cortex.c
stm32f3xx_hal_dma.c
stm32f3xx_hal_exti.c
stm32f3xx_hal_flash.c
stm32f3xx_hal_flash_ex.c
stm32f3xx_hal_gpio.c
stm32f3xx_hal_i2c.c
stm32f3xx_hal_i2c_ex.c
stm32f3xx_hal_pwr.c
stm32f3xx_hal_pwr_ex.c
stm32f3xx_hal_rcc.c
stm32f3xx_hal_rcc_ex.c
stm32f3xx_hal_rtc.c
stm32f3xx_hal_rtc_ex.c
stm32f3xx_hal_uart.c
stm32f3xx_hal_uart_ex.c

```
================================================
FILE: Kaggle/model-training.ipynb
================================================
# Jupyter notebook converted to Python script.

# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session


"""
### Loading the CSV Files
"""

normal = pd.read_csv('/kaggle/input/dc-motor/reading_log_20250626_185157.csv')
load = pd.read_csv('/kaggle/input/dc-motor/reading_log_20250626_185723.csv')
random = pd.read_csv('/kaggle/input/dc-motor/reading_log_20250626_190021.csv')

"""
### Parameters

"""

numSamples = 5000
shifting = 5
columnNumber = 1
shiftRange = 200

"""
### Dataset Generation
"""

import numpy as np
import pandas as pd

def generate(signal_df: pd.DataFrame, label: int,
                                segment_length: int = 2000,
                                num_segments: int = numSamples,
```

```
                                    shifts_per_segment: int = shifting,
                                    shift_range:    tuple    =    (-shiftRange,
shiftRange)) -> pd.DataFrame:

    signal = signal_df.iloc[:, columnNumber].values
    max_start = len(signal) - segment_length

    samples = []

    for _ in range(num_segments):
        start_idx = np.random.randint(0, max_start)
        segment = signal[start_idx:start_idx + segment_length]

        unshifted = np.append(segment, label)
        samples.append(unshifted)

        for _ in range(shifts_per_segment):
            shift = np.random.uniform(*shift_range)
            shifted_segment = segment + shift
            shifted = np.append(shifted_segment, label)
            samples.append(shifted)

    columns = [f"v{i}" for i in range(segment_length)] + ["label"]
    return pd.DataFrame(samples, columns=columns)


normal = generate(normal, label=0)
load = generate(load, label=0)
random_1 = generate(random, label=1)
random_2 = generate(random, label=1)

"""
### Train and Test Dataset Split
"""

from sklearn.model_selection import train_test_split

combined_df = pd.concat([normal, load, random_1, random_2], ignore_index=True)

train_df, test_df = train_test_split(
    combined_df,
    test_size=0.2,
    stratify=combined_df['label'],
    random_state=42
)

print("Train label counts:\n", train_df['label'].value_counts())
print("Test label counts:\n", test_df['label'].value_counts())


"""
### Plot few samples from Train and Test Dataset
"""

num = 5

import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

def plot_individual_samples(df, dataset_name, column_start=0, column_end=-1):
    sample_df = df.sample(n=num, random_state=42).reset_index(drop=True)

    for idx in range(num):
        row = sample_df.iloc[idx]
        label = row['label']

        if column_end == -1:
            signal = row.iloc[column_start:-1].values.astype(float)
        else:
            signal = row.iloc[column_start:column_end].values.astype(float)

        plt.figure(figsize=(8, 3))
        plt.plot(signal)
        plt.title(f"{dataset_name} Sample {idx+1} - Label: {label}")
        plt.xlabel("Time")
        plt.ylabel("Amplitude")
        plt.grid(True)
        plt.tight_layout()
        plt.show()

plot_individual_samples(train_df, dataset_name="Train")
plot_individual_samples(test_df, dataset_name="Test")


"""
# ResNet
"""

"""
### Model Definition and Training
"""

import pandas as pd
import numpy as np
import torch
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from torch.utils.data import TensorDataset, DataLoader
import torch.nn.functional as F
import torch.nn as nn
import matplotlib.pyplot as plt
import random

df = train_df
X = df.drop('label', axis=1).values
y = df['label'].values.astype(int)

X_dev, X_test, y_dev, y_test = train_test_split(X, y, test_size=100,
stratify=y, random_state=42)

scaler = StandardScaler()
X_dev_scaled = scaler.fit_transform(X_dev)
```

```
X_test_scaled = scaler.transform(X_test)

X_dev_scaled = X_dev_scaled.reshape(-1, 1, 40, 50)
X_test_scaled = X_test_scaled.reshape(-1, 1, 40, 50)

X_train, X_val, y_train, y_val = train_test_split(X_dev_scaled, y_dev,
test_size=0.2, stratify=y_dev, random_state=42)

class SignalDataset(torch.utils.data.Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X).float()
        self.y = torch.tensor(y).long()

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

train_ds = SignalDataset(X_train, y_train)
val_ds = SignalDataset(X_val, y_val)
test_ds = SignalDataset(X_test_scaled, y_test)

train_loader = DataLoader(train_ds, batch_size=50, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=50)
test_loader = DataLoader(test_ds, batch_size=50)

class BasicBlock(nn.Module):
    def __init__(self, in_channels, dropout=0.3):
        super().__init__()
        self.conv_block = nn.Sequential(
            nn.Conv2d(in_channels, in_channels, 3, padding=1),
            nn.BatchNorm2d(in_channels),
            nn.ReLU(),
            nn.Conv2d(in_channels, in_channels, 3, padding=1),
            nn.BatchNorm2d(in_channels)
        )
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout2d(dropout)

    def forward(self, x):
        out = self.conv_block(x)
        out = self.dropout(out)
        return self.relu(x + out)

class ResNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.initial = nn.Sequential(
            nn.Conv2d(1, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU()
        )
        self.layer1 = BasicBlock(64)
        self.layer2 = BasicBlock(64)
        self.layer3 = BasicBlock(64)
        self.fc = nn.Sequential(
```

```
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Linear(64, 2)
        )

    def forward(self, x):
        x = self.initial(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        return self.fc(x)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = ResNet().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
scheduler   =   torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,   'max',
patience=5, factor=0.5)

def evaluate(model, loader):
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for xb, yb in loader:
            xb, yb = xb.to(device), yb.to(device)
            out = model(xb)
            correct += (out.argmax(1) == yb).sum().item()
            total += yb.size(0)
    return correct / total

def train_model(model, train_loader, val_loader, epochs=100, patience=10):
    best_acc = 0
    wait = 0
    for epoch in range(epochs):
        model.train()
        total_loss, correct, total = 0, 0, 0
        for xb, yb in train_loader:
            xb, yb = xb.to(device), yb.to(device)
            optimizer.zero_grad()
            out = model(xb)
            loss = criterion(out, yb)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
            correct += (out.argmax(1) == yb).sum().item()
            total += yb.size(0)

        val_acc = evaluate(model, val_loader)
        scheduler.step(val_acc)

        print(f"Epoch {epoch+1:02d} | Loss: {total_loss/len(train_loader):.4f}
| Train Acc: {correct/total:.4f} | Val Acc: {val_acc:.4f}")

        if val_acc > best_acc:
            best_acc = val_acc
            best_model = model.state_dict()
            wait = 0
```

```
        else:
            wait += 1
            if wait >= patience:
                print("Early stopping triggered.")
                break

    model.load_state_dict(best_model)

train_model(model, train_loader, val_loader)

test_acc = evaluate(model, test_loader)
print(f"\nFinal Test Accuracy: {test_acc:.4f}")


"""
### Testing on Test Dataset
"""

from sklearn.metrics import accuracy_score, precision_score, recall_score

new_test_df = test_df
X_new_test = new_test_df.drop('label', axis=1).values
y_true = new_test_df['label']

X_new_test_scaled = scaler.transform(X_new_test)
X_new_test_scaled = X_new_test_scaled.reshape(-1, 1, 40, 50)

model.eval()
new_test_ds = TensorDataset(torch.tensor(X_new_test_scaled).float())
new_test_loader = DataLoader(new_test_ds, batch_size=50)

new_preds = []
with torch.no_grad():
    for xb in new_test_loader:
        xb = xb[0].to(device)
        preds = model(xb).argmax(dim=1)
        new_preds.extend(preds.cpu().numpy())

acc = accuracy_score(y_true, new_preds)
prec = precision_score(y_true, new_preds)
rec = recall_score(y_true, new_preds)

print(f"\nEvaluation on test dataframe:")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")



"""
### Save the Model
"""

import joblib

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
model.to(device)
model.eval()

dummy_input = torch.randn(1, 1, 40, 50).to(device)

traced_model = torch.jit.trace(model, dummy_input)
traced_model.save('model.pt')
joblib.dump(scaler, 'scaker.pkl')

"""
# CNN
"""

class CNN2D(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.AdaptiveAvgPool2d(1)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128, 2)
        )

    def forward(self, x):
        x = self.conv(x)
        return self.classifier(x)


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = CNN2D().to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
scheduler   =   torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,   'max',
patience=5, factor=0.5)

train_model(model, train_loader, val_loader)
test_acc = evaluate(model, test_loader)
print(f"Test Accuracy: {test_acc:.4f}")


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
model.to(device)
model.eval()

dummy_input = torch.randn(1, 1, 40, 50).to(device)
traced_model = torch.jit.trace(model, dummy_input)

traced_model.save("CNN_2D.pt")

"""
# U-Net
"""

import torch
import torch.nn as nn
import torch.nn.functional as F

def center_crop(enc_feat, target_feat):
    _, _, h, w = enc_feat.shape
    _, _, th, tw = target_feat.shape
    dh, dw = (h - th) // 2, (w - tw) // 2
    return enc_feat[:, :, dh:dh+th, dw:dw+tw]

class UNet2D(nn.Module):
    def __init__(self):
        super(UNet2D, self).__init__()

        self.enc1 = self.conv_block(1, 64)
        self.pool1 = nn.MaxPool2d(2)

        self.enc2 = self.conv_block(64, 128)
        self.pool2 = nn.MaxPool2d(2)

        self.bottleneck = self.conv_block(128, 256)

        self.up2 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2)
        self.dec2 = self.conv_block(256, 128)

        self.up1 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)
        self.dec1 = self.conv_block(128, 64)

        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d((1, 1)),
            nn.Flatten(),
            nn.Linear(64, 2)
        )

    def conv_block(self, in_channels, out_channels):
        return nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, 3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )
```

```
    def forward(self, x):
        enc1 = self.enc1(x)
        enc2 = self.enc2(self.pool1(enc1))
        bottleneck = self.bottleneck(self.pool2(enc2))

        up2 = self.up2(bottleneck)
        enc2 = center_crop(enc2, up2)
        dec2 = self.dec2(torch.cat([up2, enc2], dim=1))

        up1 = self.up1(dec2)
        enc1 = center_crop(enc1, up1)
        dec1 = self.dec1(torch.cat([up1, enc1], dim=1))

        return self.classifier(dec1)


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = UNet2D().to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
scheduler   =   torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,   'max',
patience=5, factor=0.5)

train_model(model, train_loader, val_loader)
test_acc = evaluate(model, test_loader)
print(f"Test Accuracy: {test_acc:.4f}")


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model.to(device)
model.eval()

dummy_input = torch.randn(1, 1, 40, 50).to(device)
traced_model = torch.jit.trace(model, dummy_input)

traced_model.save("UNet.pt")

"""
# VGG
"""

class VGG2D(nn.Module):
    def __init__(self):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 64, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 64, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 128, 3, padding=1),
```

```
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(128, 256, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.ReLU(),
            nn.AdaptiveAvgPool2d(1)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256, 2)
        )

    def forward(self, x):
        x = self.features(x)
        return self.classifier(x)


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = VGG2D().to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
scheduler   =   torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,   'max',
patience=5, factor=0.5)

train_model(model, train_loader, val_loader)
test_acc = evaluate(model, test_loader)
print(f"Test Accuracy: {test_acc:.4f}")


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model.to(device)
model.eval()

dummy_input = torch.randn(1, 1, 40, 50).to(device)
traced_model = torch.jit.trace(model, dummy_input)

traced_model.save("VGG_2D.pt")

"""
# MLP
"""

import torch
import torch.nn as nn

class MLP2D(nn.Module):
    def __init__(self, input_shape=(1, 40, 50), hidden_dim=128, output_dim=2):
        super().__init__()
        self.flatten = nn.Flatten()
        input_dim = input_shape[0] * input_shape[1] * input_shape[2]  # 1*40*50
= 2000
        self.classifier = nn.Sequential(
```

```
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(hidden_dim, output_dim)  # 2 classes
        )

    def forward(self, x):
        x = self.flatten(x)
        return self.classifier(x)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = MLP2D().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
scheduler   =   torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,   'max',
patience=5, factor=0.5)

train_model(model, train_loader, val_loader)

test_acc = evaluate(model, test_loader)
print(f"Test Accuracy: {test_acc:.4f}")


dummy_input = torch.randn(1, 1, 40, 50).to(device)
traced = torch.jit.trace(model, dummy_input)
traced.save("MLP_2D.pt")



================================================
FILE: Raspberry Pi/app.py
================================================
import threading
import time
from tkinter import *
from tkinter import ttk, messagebox
from collections import deque
import numpy as np
import matplotlib
matplotlib.use("TkAgg")
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import serial

from   inference   import   load_model_and_scaler,   inference_worker_improved,
MODEL_SCALER_MAP

WINDOW_SIZE     = 2000
SAMPLE_RATE     = 2000
UPDATE_INTERVAL = 50
#GAIN            = 0.0245   # DC-DC Boost Converter Gain
GAIN            = 0.561     # DC Motor Gain

T     = 1.0 / SAMPLE_RATE
```

```
tau   = 1.0 / 25.0
alpha = T / (T + tau)

smoothing_window = deque(maxlen=10)

try:
    ser = serial.Serial('/dev/serial0', 115200, timeout=1)
except serial.SerialException as e:
    print(f"Error opening serial port: {e}")
    messagebox.showerror("Serial Error",
                         f"Could not open serial port /dev/serial0.\n{e}")
    exit()

try:
    first = ser.readline().decode('utf-8', errors='ignore').strip()
    if first.isdigit():
        prev_filtered = [int(first) * GAIN]
    else:
        prev_filtered = [0.0]
except Exception:
    prev_filtered = [0.0]

buffer = ""

def reader_thread(shared_state, stop_event):
    global buffer
    while not stop_event.is_set():
        try:
            chunk    =    ser.read(ser.in_waiting    or    1).decode('utf-8',
errors='ignore')
            buffer += chunk

            if '\n' in buffer:
                lines = buffer.split('\n')
                buffer = lines.pop()

                out_vals = []
                for line in lines:
                    line = line.strip()
                    if not line or not line.isdigit():
                        continue

                    raw = int(line)
                    voltage = raw * GAIN

                    filtered = alpha * voltage + (1 - alpha) * prev_filtered[0]
                    prev_filtered[0] = filtered

                    smoothing_window.append(filtered)
                    smoothed = sum(smoothing_window) / len(smoothing_window)

                    out_vals.append(smoothed)

                if out_vals:
                    with shared_state['lock']:
                        shared_state['queue'].extend(out_vals)
```

```
        except Exception as e:
            print(f"[Reader] {e}")
        time.sleep(0.001)

class RealTimeMonitor:
    def __init__(self, root, shared_state):
        self.root = root
        self.shared_state = shared_state
        self.setup_gui()
        self.update_gui()

    def setup_gui(self):
        self.root.title("Real-Time Voltage Monitor")
        self.root.geometry("900x600")
        self.root.configure(bg='#f0f2f5')

        cf = Frame(self.root, bg='#f0f2f5', pady=5)
        cf.pack(fill=X)

        Label(cf, text="Prediction:", font=("Arial",12), bg='#f0f2f5')\
            .pack(side=LEFT, padx=(10,5))
        self.pred_var = StringVar(self.root, "Initializing…")
        Label(cf, textvariable=self.pred_var,
              font=("Arial",12,"bold"), fg="#0056b3", bg="#e7f3ff",
              padx=10, pady=5, relief="groove")\
            .pack(side=LEFT)

        self.status_var = StringVar(self.root, "Running")
        Label(cf, textvariable=self.status_var,
              font=("Arial",10,"bold"), fg="green", bg='#f0f2f5')\
            .pack(side=RIGHT, padx=10)
        Button(cf, text="Quit", command=self.quit_app,
               bg="#dc3545", fg="white", font=("Arial",10))\
            .pack(side=RIGHT, padx=5)

        pf = Frame(self.root, bg='white')
        pf.pack(fill=BOTH, expand=True, padx=10, pady=10)

        self.fig = Figure(figsize=(8,4), dpi=100)
        self.ax  = self.fig.add_subplot(111)
        x = np.arange(WINDOW_SIZE)
        self.line, = self.ax.plot(x, np.zeros(WINDOW_SIZE), lw=1.5)

        self.ax.set_xlim(0, WINDOW_SIZE)
        self.ax.set_ylim(0, 400)

        self.ax.set_title("Voltage (V)")
        self.ax.set_xlabel("Sample")
        self.ax.set_ylabel("V")
        self.ax.grid(True, linestyle='--', alpha=0.5)
        self.fig.tight_layout()

        self.canvas = FigureCanvasTkAgg(self.fig, master=pf)
        self.canvas.draw()
        self.canvas.get_tk_widget().pack(fill=BOTH, expand=True)

    def update_gui(self):
```

```
        try:
            with self.shared_state['lock']:
                data = list(self.shared_state['queue'])
                pred = self.shared_state.get('prediction', "N/A")
                pc   = self.shared_state.get('prediction_class', 0)

            self.pred_var.set(pred)
            self.line.set_color(['blue', 'red'][pc])

            if len(data) == WINDOW_SIZE:
                self.line.set_ydata(data)

                y_min = min(data)
                y_max = max(data)
                margin = (y_max - y_min) * 0.1 if y_max != y_min else 10
                self.ax.set_ylim(y_min - margin, y_max + margin)

            self.canvas.draw_idle()
        except Exception as e:
            print(f"[GUI] {e}")

        self.root.after(UPDATE_INTERVAL, self.update_gui)


    def quit_app(self):
        self.status_var.set("Shutting down…")
        self.root.after(100, self.root.quit)

def main():
    sel_root = Tk()
    sel_root.title("Select Model")
    model_var = StringVar(sel_root, "ResNet-Boost")

    def on_start():
        sel_root.destroy()

    Label(sel_root, text="Select a model:", font=("Arial",12)).pack(pady=10)
    cb = ttk.Combobox(sel_root, textvariable=model_var,
                      values=list(MODEL_SCALER_MAP.keys()),
                      state="readonly")
    cb.pack(pady=5, padx=10)
    Button(sel_root, text="Start", command=on_start).pack(pady=10)
    sel_root.mainloop()

    model_name = model_var.get()
    model, scaler = load_model_and_scaler(model_name)
    if model is None or scaler is None:
        messagebox.showerror("Error", f"Could not load '{model_name}'")
        return

    shared_state = {
        "queue": deque([0.0]*WINDOW_SIZE, maxlen=WINDOW_SIZE),
        "lock": threading.Lock(),
        "prediction": "Initializing…",
        "prediction_class": 0
    }
    stop_evt = threading.Event()
```

```
    threading.Thread(target=reader_thread,
                     args=(shared_state, stop_evt),
                     daemon=True).start()
    threading.Thread(target=inference_worker_improved,
                     args=(model, scaler, shared_state, stop_evt),
                     daemon=True).start()

    root = Tk()
    app = RealTimeMonitor(root, shared_state)

    def on_close():
        stop_evt.set()
        time.sleep(0.1)
        ser.close()
        root.destroy()

    root.protocol("WM_DELETE_WINDOW", on_close)
    root.mainloop()

if __name__ == "__main__":
    main()



================================================
FILE: Raspberry Pi/inference.py
================================================
import torch
import numpy as np
import joblib
import time
import csv
from collections import deque

WINDOW_SIZE = 2000
INFERENCE_INTERVAL = 1

# Change the path of the models as per the new directory structure
MODEL_SCALER_MAP = {
    "ResNet-Motor": ("/home/pi/project_with_plot_tkinter_uart/motor/model.pt",
"/home/pi/project_with_plot_tkinter_uart/motor/scaler.pkl"),
    "ResNet-Boost":
("/home/pi/project_with_plot_tkinter/model_1/resnet40_50.pt",
"/home/pi/project_with_plot_tkinter/model_1/standard_scaler_40_50.pkl"),
    "CNN-Boost":      ("/home/pi/project_with_plot_tkinter/model_2/CNN_2D.pt",
"/home/pi/project_with_plot_tkinter/model_2/standard_scaler_40_50.pkl"),
    "VGG-Boost":         ("/home/pi/project_with_plot_tkinter/model_3/VGG.pt",
"/home/pi/project_with_plot_tkinter/model_3/standard_scaler_40_50.pkl"),
    "U-Net-Boost":
("/home/pi/project_with_plot_tkinter/model_4/UNet2D_40x50.pt",
"/home/pi/project_with_plot_tkinter/model_4/standard_scaler_40_50.pkl"),
    "MLP-Boost":      ("/home/pi/project_with_plot_tkinter/model_5/MLP_2D.pt",
"/home/pi/project_with_plot_tkinter/model_5/standard_scaler_40_50.pkl")
}

def load_model_and_scaler(model_choice):
```

```
    try:
        model_path, scaler_path = MODEL_SCALER_MAP[model_choice]
        model = torch.jit.load(model_path, map_location='cpu')
        model.eval()
        scaler = joblib.load(scaler_path)
        print(f"Loaded {model_choice} successfully.")
        return model, scaler
    except Exception as e:
        print(f"Error loading model or scaler for {model_choice}: {e}")
        return None, None

def inference_worker_improved(model, scaler, shared_state, stop_event):
    from datetime import datetime

    label_map = {0: "Normal", 1: "Abnormal"}
    history = deque(maxlen=2)

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    volt_log_file = open(f"reading_log_{timestamp}.csv", "w", newline='')
    pred_log_file = open(f"prediction_log_{timestamp}.csv", "w", newline='')

    volt_writer = csv.writer(volt_log_file)
    volt_writer.writerow(["Timestamp", "Voltage"])

    pred_writer = csv.writer(pred_log_file)
    pred_writer.writerow(["Timestamp", "Prediction"])

    while not stop_event.is_set():
        time.sleep(INFERENCE_INTERVAL)

        try:
            with shared_state['lock']:
                data = np.array(shared_state['queue'])

            now = datetime.now().isoformat()
            for v in data:
                volt_writer.writerow([now, round(v, 3)])
            volt_log_file.flush()

            scaled = scaler.transform(data.reshape(1, -1))
            input_tensor                =                torch.tensor(scaled,
dtype=torch.float32).reshape(1, 1, 40, 50)

            with torch.no_grad():
                output = model(input_tensor)
                pred_idx = torch.argmax(output, dim=1).item()

            history.append(pred_idx)

            if list(history) == [1, 1]:
                final = 1
            else:
                final = 0

            prediction = label_map[final]

            with shared_state['lock']:
```

```
                shared_state['prediction'] = prediction
                shared_state['prediction_class'] = final

            pred_writer.writerow([now, prediction])
            pred_log_file.flush()

            print(f"🔍 Prediction: {prediction}")

        except Exception as e:
            print(f"Inference error: {e}")
            with shared_state['lock']:
                shared_state['prediction'] = "Error"
                shared_state['prediction_class'] = 1
```

```
================================================
FILE: Raspberry Pi/plot.py
================================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

Column_Number = 1

df = pd.read_csv('/home/pi/adc_filtered_data.csv')

row = df.iloc[:,Column_Number]

plt.plot(row)
plt.show()
```

```
================================================
FILE: Raspberry Pi/requirements.txt
================================================
blinker==1.9.0
click==8.2.1
contourpy==1.3.2
cycler==0.12.1
Flask==3.1.1
fonttools==4.58.4
itsdangerous==2.2.0
Jinja2==3.1.6
kiwisolver==1.4.8
MarkupSafe==3.0.2
matplotlib==3.10.3
numpy==2.3.1
packaging==25.0
pandas==2.3.0
pillow==11.2.1
pyparsing==3.2.3
pyserial==3.5
python-dateutil==2.9.0.post0
pytz==2025.2
six==1.17.0
```

```
tzdata==2025.2
Werkzeug==3.1.3
blinker==1.9.0
click==8.2.1
contourpy==1.3.2
cycler==0.12.1
filelock==3.18.0
Flask==3.1.1
fonttools==4.58.4
fsspec==2025.5.1
itsdangerous==2.2.0
Jinja2==3.1.6
kiwisolver==1.4.8
MarkupSafe==3.0.2
matplotlib==3.10.3
mpmath==1.3.0
networkx==3.5
numpy==2.3.1
packaging==25.0
pandas==2.3.0
pillow==11.2.1
pyparsing==3.2.3
pyserial==3.5
python-dateutil==2.9.0.post0
pytz==2025.2
six==1.17.0
sympy==1.14.0
torch==2.7.1
typing_extensions==4.14.0
tzdata==2025.2
Werkzeug==3.1.3
blinker==1.9.0
click==8.2.1
contourpy==1.3.2
cycler==0.12.1
filelock==3.18.0
Flask==3.1.1
fonttools==4.58.4
fsspec==2025.5.1
itsdangerous==2.2.0
Jinja2==3.1.6
joblib==1.5.1
kiwisolver==1.4.8
MarkupSafe==3.0.2
matplotlib==3.10.3
mpmath==1.3.0
networkx==3.5
numpy==2.3.1
packaging==25.0
pandas==2.3.0
pillow==11.2.1
pyparsing==3.2.3
pyserial==3.5
python-dateutil==2.9.0.post0
pytz==2025.2
six==1.17.0
sympy==1.14.0
```

```
tk==0.1.0
torch==2.7.1
typing_extensions==4.14.0
tzdata==2025.2
Werkzeug==3.1.3
blinker==1.9.0
click==8.2.1
contourpy==1.3.2
cycler==0.12.1
filelock==3.18.0
Flask==3.1.1
fonttools==4.58.4
fsspec==2025.5.1
itsdangerous==2.2.0
Jinja2==3.1.6
joblib==1.5.1
kiwisolver==1.4.8
MarkupSafe==3.0.2
matplotlib==3.10.3
mpmath==1.3.0
networkx==3.5
numpy==2.3.1
packaging==25.0
pandas==2.3.0
pillow==11.2.1
pyparsing==3.2.3
pyserial==3.5
python-dateutil==2.9.0.post0
pytz==2025.2
scikit-learn==1.7.0
scipy==1.16.0
six==1.17.0
sympy==1.14.0
threadpoolctl==3.6.0
tk==0.1.0
torch==2.7.1
typing_extensions==4.14.0
tzdata==2025.2
Werkzeug==3.1.3
blinker==1.9.0
click==8.2.1
contourpy==1.3.2
cycler==0.12.1
filelock==3.18.0
Flask==3.1.1
fonttools==4.58.4
fsspec==2025.5.1
itsdangerous==2.2.0
Jinja2==3.1.6
joblib==1.5.1
kiwisolver==1.4.8
MarkupSafe==3.0.2
matplotlib==3.10.3
mpmath==1.3.0
networkx==3.5
numpy==2.3.1
packaging==25.0
```

```
pandas==2.3.0
pillow==11.2.1
pyparsing==3.2.3
pyserial==3.5
python-dateutil==2.9.0.post0
pytz==2025.2
scikit-learn==1.7.0
scipy==1.16.0
six==1.17.0
sympy==1.14.0
threadpoolctl==3.6.0
tk==0.1.0
torch==2.7.1
typing_extensions==4.14.0
tzdata==2025.2
Werkzeug==3.1.3
```

```
================================================
FILE: Raspberry Pi/uart.py
================================================
import serial
import time
from collections import deque
from datetime import datetime

ser = serial.Serial('/dev/serial0', 115200, timeout=1)

print("Receiving and filtering ADC data from STM32...\n")

sample_count = 0
start_time = time.time()

T = 0.0005
tau = 1 / 25
alpha = T / (T + tau)
gain = 0.561

prev_filtered = 0
smoothing_window = deque(maxlen=10)

try:
    while True:
        try:
            line = ser.readline().decode('utf-8').strip()
        except UnicodeDecodeError:
            continue

        if line.isdigit():
            adc_val = int(line) * gain
            filtered_val = alpha * adc_val + (1 - alpha) * prev_filtered
            prev_filtered = filtered_val

            print(f'Raw: {adc_val:.2f}, Filtered: {filtered_val:.2f}')

            sample_count += 1
```

```
        current_time = time.time()
        if current_time - start_time >= 1.0:
            print(f"Samples per second: {sample_count}")
            sample_count = 0
            start_time = current_time

except KeyboardInterrupt:
    print("\nStopped by user.")
finally:
    ser.close()
```

```
================================================
FILE: Raspberry Pi/model_1/resnet40_50.pt
================================================
[Binary file]


================================================
FILE: Raspberry Pi/model_1/standard_scaler_40_50.pkl
================================================
[Binary file]


================================================
FILE: Raspberry Pi/model_2/CNN_2D.pt
================================================
[Binary file]


================================================
FILE: Raspberry Pi/model_2/standard_scaler_40_50.pkl
================================================
[Binary file]


================================================
FILE: Raspberry Pi/model_3/standard_scaler_40_50.pkl
================================================
[Binary file]


================================================
FILE: Raspberry Pi/model_3/VGG.pt
================================================
[Binary file]


================================================
FILE: Raspberry Pi/model_4/standard_scaler_40_50.pkl
================================================
[Binary file]


================================================
```

FILE: Raspberry Pi/model_4/UNet2D_40x50.pt
================================================
[Binary file]


================================================
FILE: Raspberry Pi/model_5/MLP_2D.pt
================================================
[Binary file]


================================================
FILE: Raspberry Pi/model_5/standard_scaler_40_50.pkl
================================================
[Binary file]


================================================
FILE: Raspberry Pi/model_6/model.pt
================================================
[Binary file]


================================================
FILE: Raspberry Pi/model_6/model_ResNet.pt
================================================
[Binary file]


================================================
FILE: Raspberry Pi/model_6/scaker.pkl
================================================
[Binary file]


================================================
FILE: Raspberry Pi/model_6/scaler.pkl
================================================
[Binary file]

================================================
FILE: STM32/main.h
================================================

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
*
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
*
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
```

```
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *

******************************************************************************
*
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------
*/
#include "main.h"
#include <stdio.h>
#include <string.h>

/* Private includes ----------------------------------------------------------
*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------
*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------
*/
/* USER CODE BEGIN PD */
#define COEFF_B0 (1.0f / 18.0f)
#define COEFF_B1 (1.0f / 18.0f)
#define COEFF_A1 (8.0f / 9.0f)

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------
*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------
*/
ADC_HandleTypeDef hadc1;

RTC_HandleTypeDef hrtc;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */
static float prev_input = 0.0f;
static float prev_output = 0.0f;

/* USER CODE END PV */
```

```
/* Private function prototypes -----------------------------------------------
*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_RTC_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART1_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------
*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------
*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick.
*/
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_RTC_Init();
  MX_ADC1_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */
  char msg[30];// Buffer to store message
  uint16_t adc_val = 0; // Raw ADC value variable
  uint16_t filtered_val_tx = 0; // Filtered value to be transmitted
  float current_filtered_val = 0.0f; // ADC value variable
```

```
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        /* NOTE: For a precise 1700 Hz sampling rate, a hardware timer should
be used
         * to trigger the ADC conversion. Polling in a while-loop as done here
will
         * result in timing
         */
  HAL_ADC_Start(&hadc1);
  HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
  adc_val = HAL_ADC_GetValue(&hadc1);
  HAL_ADC_Stop(&hadc1);

        /* Apply the IIR low-pass filter */
  current_filtered_val = (COEFF_B0 * adc_val) + (COEFF_B1 * prev_input) +
(COEFF_A1 * prev_output);

        /* Update state variables for the next iteration */
  prev_input = (float)adc_val;
  prev_output = current_filtered_val;

        /* Prepare the filtered value for transmission */
  filtered_val_tx = (uint16_t)current_filtered_val;

  sprintf(msg, "%d\r\n", filtered_val_tx); // Format filtered data
  HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);

  }

  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType                                        =
RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
```

```
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_RTC
                              |RCC_PERIPHCLK_ADC12;
  PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
  PeriphClkInit.Adc12ClockSelection = RCC_ADC12PLLCLK_DIV1;
  PeriphClkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSI;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_MultiModeTypeDef multimode = {0};
  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
```

```
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc1.Init.LowPowerAutoWait = DISABLE;
  hadc1.Init.Overrun = ADC_OVR_DATA_OVERWRITTEN;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure the ADC multi-mode
  */
  multimode.Mode = ADC_MODE_INDEPENDENT;
  if (HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_5;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SingleDiff = ADC_SINGLE_ENDED;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  sConfig.OffsetNumber = ADC_OFFSET_NONE;
  sConfig.Offset = 0;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief RTC Initialization Function
  * @param None
  * @retval None
  */
static void MX_RTC_Init(void)
{

  /* USER CODE BEGIN RTC_Init 0 */

  /* USER CODE END RTC_Init 0 */

  /* USER CODE BEGIN RTC_Init 1 */

  /* USER CODE END RTC_Init 1 */
```

```
  /** Initialize RTC Only
  */
  hrtc.Instance = RTC;
  hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
  hrtc.Init.AsynchPrediv = 127;
  hrtc.Init.SynchPrediv = 255;
  hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
  hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
  hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN RTC_Init 2 */

  /* USER CODE END RTC_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
```

```
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  /* USER CODE BEGIN MX_GPIO_Init_1 */

  /* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();

  /* USER CODE BEGIN MX_GPIO_Init_2 */

  /* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state
*/
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

# 1. Data Collection from STM32

- **Hardware Used:** STM32 Nucleo-F303ZE
- **Signal Acquisition:**
  - **ADC Input:** PF4 (ADC1_IN5) is used to read the analog signal.
  - **UART Output:** PC4 (USART2_TX) and PC5 (USART2_RX) are used for UART communication with the Raspberry Pi.
- **Firmware:** The STM32 firmware reads ADC values, applies a digital filter, and transmits filtered values over UART at 115200 baud.

**Pin Connections:**

- STM32 PF4: Connect to the analog signal source (e.g., output of Boost Converter).
- STM32 PC4 (TX): Connect to Raspberry Pi RX (GPIO15, physical pin 10).
- STM32 PC5 (RX): Connect to Raspberry Pi TX (GPIO14, physical pin 8).
- Common GND between STM32 and Raspberry Pi.

---

# 2. Uploading Data to Kaggle

1. **Collect Data:**
   - Use the Raspberry Pi to receive UART data and log it as CSV files (see `Raspberry Pi/uart.py`).
   - Example files: `normal.csv`, `random.csv`, `load.csv` in `Dataset/DC Motor/` or `Dataset/Boost Converter/`.
2. **Upload to Kaggle:**
   - Go to Kaggle Datasets.
   - Click "New Dataset" and upload your CSV files.
   - Fill in the dataset details and make it public or private as needed.

---

# 3. Model Training on Kaggle

1. **Open the Notebook:**
   - Use `Kaggle/model-training.ipynb` as your starting point.

2. **Load the Dataset:**
    - Use Kaggle's data path, e.g.:

```
import pandas as pd
normal = pd.read_csv('/kaggle/input/your-dataset/normal.csv')
load = pd.read_csv('/kaggle/input/your-dataset/load.csv')
random = pd.read_csv('/kaggle/input/your-dataset/random.csv')
```

3. **Preprocess and Generate Training Data:**
    - The notebook provides functions to segment, label, and augment the data.
4. **Activate GPU:**
    - In the Kaggle notebook, go to `Settings` (right sidebar) and set "Accelerator" to "GPU".
5. **Train the Model:**
    - The notebook includes code for training several models (ResNet, CNN, VGG, U-Net, MLP).
    - Training uses PyTorch and scikit-learn.
6. **Save the Model and Scaler:**
    - After training, save the model and scaler:

```
import torch
import joblib
traced_model = torch.jit.trace(model, dummy_input)
traced_model.save('model.pt')
joblib.dump(scaler, 'scaler.pkl')
```

    - Download these files from the Kaggle notebook output.

---

# 4. Deploying to Raspberry Pi

1. **Setup Raspberry Pi:**
    - Install dependencies:

```
pip install -r Raspberry\ Pi/requirements.txt
```

    - Ensure `pyserial`, `torch`, `joblib`, `matplotlib`, `tk`, etc. are installed.

2. **Copy Model Files:**
    - Create a new folder in `Raspberry Pi/` (e.g., `model_1/` ).
    - Add your `model.pt` and `scaler.pkl` to this folder.
3. **Update Model Paths:**
    - Edit `Raspberry Pi/inference.py` and update `MODEL_SCALER_MAP` with your new model folder and file names.

---

# 5. Running Real-Time Inference

- Run the main application:

```
python Raspberry\ Pi/app.py
```

- The app:
    - Reads UART data from STM32.
    - Applies the trained model for anomaly detection.
    - Displays real-time plots and predictions.
    - **Records all incoming data and predictions to CSV logs.**
- **Retraining:** If new data is collected, you can upload it to Kaggle and repeat the training process.

---

# 6. Notes on Data Logging and Retraining

- Every run of the inference app logs both the raw voltage and the model's predictions.
- These logs can be used to further improve or retrain your models.

---

# 7. Hardware Pin Summary

| Function | STM32 Pin | STM32 Peripheral | Raspberry Pi Pin |
|---|---|---|---|
| ADC Input | PF4 | ADC1_IN5 | Analog Signal |
| UART TX (to Pi) | PC4 | USART2_TX | Pin:8 (RX) |
| UART RX (from Pi) | PC5 | USART2_RX | Pin:10 (TX) |
| GND | GND | - | GND |

---

# 8. Troubleshooting

- **Serial Port:** Ensure `/dev/serial0` is enabled on Raspberry Pi ( `raspi-config` > Interface Options > Serial).
- **Baud Rate:** Both STM32 and Pi must use 115200 baud.
- **Permissions:** You may need to add your user to the `dialout` group on the Pi for serial access.

---

# 9. References


- STM32 Nucleo-F303ZE Datasheet
- Kaggle Documentation
- PyTorch Documentation