Detección de Anomalías en Motores usando TinyML en Edge

Bautista Mercado
Facultad de Informática, UNLP
Internet de las Cosas

Julio 2025

Índice

1. Introducción

En diversos entornos productivos, como talleres, invernaderos, instalaciones rurales o sistemas de ventilación, o hasta incluso entornos industriales es habitual encontrar motores funcionando de forma continua. Estos motores, a pesar de ser críticos para el funcionamiento general del sistema, muchas veces carecen de monitoreo adecuado.

Detectar fallas de funcionamiento, vibraciones anómalas o comportamientos fuera de lo común de forma temprana puede evitar problemas mayores, desde daños materiales hasta paradas no planificadas. Sin embargo, hacerlo manualmente implica presencia constante y experiencia técnica.

Este proyecto tiene como objetivo diseñar un sistema basado en un microcontrolador ESP32 que, mediante sensores (acelerómetro y micrófono), pueda clasificar el estado de un motor en tiempo real usando machine learning embebido (TinyML). Para eesto, se utilizó la plataforma Edge Impulse, que permite entrenar y desplegar modelos de Machine Learning en dispositivos de bajo consumo.

Se busca lograr una solución compacta, económica, autónoma y que no dependa de conexión a internet, orientada a escenarios donde la supervisión constante no es viable y el monitoreo tradicional resulta costoso o inviable.

2. Planteo del problema

La mayoría de los sistemas de monitoreo de motores están orientados a entornos industriales complejos, con infraestructura disponible para adquirir, transmitir y procesar grandes volúmenes de datos. Sin embargo, en contextos más simples o económicamente limitados, los motores suelen operar sin ningún tipo de supervisión activa.

Esto implica que cualquier falla, vibración anómala o funcionamiento fuera de los parámetros normales puede pasar desapercibido hasta que provoca un desperfecto mayor. En muchos casos, la detección temprana no ocurre por falta de personal especializado, sensores adecuados o conectividad permanente.

El problema concreto es: ¨ Es posible detectar el estado de funcionamiento de un motor —normal, anómalo o apagado— sin depender de infraestructura compleja, conectividad constante ni alto costo?¨

Este proyecto aborda esto proponiendo una solución basada en hardware de bajo costo, TinyML y sensores simples pero efectivos, todo operando de forma local y autónoma.

3. Objetivos

Objetivo general:

Diseñar e implementar un sistema de detección de anomalías en motores utilizando sensores y un modelo de machine learning, capaz de clasificar en tiempo real el estado de funcionamiento de un motor sin necesidad de conectividad externa ni infraestructura compleja.

Objetivos específicos:

- Recolectar datos de funcionamiento de un motor utilizando un acelerómetro y un micrófono conectados a un ESP32.

- Etiquetar y cargar las muestras en Edge Impulse para su procesamiento y entrenamiento.
- Diseñar un modelo de clasificación capaz de distinguir entre estados: apagado, funcionamiento normal y funcionamiento anómalo.
- Integrar el modelo entrenado en el ESP32 para ejecutar inferencias en tiempo real.
- Evaluar el rendimiento del sistema en escenarios reales de prueba.
- Documentar el proceso y proponer mejoras o ampliaciones futuras del sistema.

4. Tecnologías y herramientas utilizadas

A continuación se detallan las principales tecnologías y herramientas utilizadas para el desarrollo del proyecto:

- **ESP32:** Microcontrolador de bajo costo con conectividad WiFi y Bluetooth, ideal para aplicaciones embebidas. Se usó como nodo central del sistema, ejecutando el modelo de inferencia y gestionando los sensores.
- **MPU6050:** Sensor de aceleración y giroscopio de 6 ejes. Proporcionó los datos de vibración del motor en los ejes X, Y y Z.
- **SparkFun Sound Detector:** Sensor de sonido analógico utilizado para capturar la envolvente de la señal de audio generada por el motor durante su funcionamiento.
- **Edge Impulse:** Plataforma para desarrollo de modelos de aprendizaje automático embebido. Permité recolectar datos, procesarlos, entrenar modelos, optimizarlos e integrarlos en dispositivos como el ESP32.
- **Arduino IDE:** Entorno de desarrollo utilizado para programar el ESP32, tanto para la recolección de muestras como para la ejecución de inferencias.
- **edge-impulse-data-forwarder:** Herramienta de Edge Impulse que permite enviar datos desde el monitor serie (donde el ESP32 va imprimiendo los datos sensados) directamente a la plataforma, facilitando el etiquetado y almacenamiento de muestras.

5. Diseño del sistema

5.1. Arquitectura general

El sistema fue diseñado con una arquitectura simple pero funcional, basada en la adquisición de datos mediante sensores conectados a un microcontrolador ESP32, el procesamiento local de estos datos mediante un modelo de aprendizaje automático embebido, y la salida del resultado de la inferencia a través del monitor serie.
En el esquema se observa la conexión entre los sensores MPU6050 y Sound Detector hacia el ESP32. Los datos sensados alimentan el modelo previamente entrenado, que ejecuta una clasificación en tiempo real y entrega un resultado textual por monitor serie.

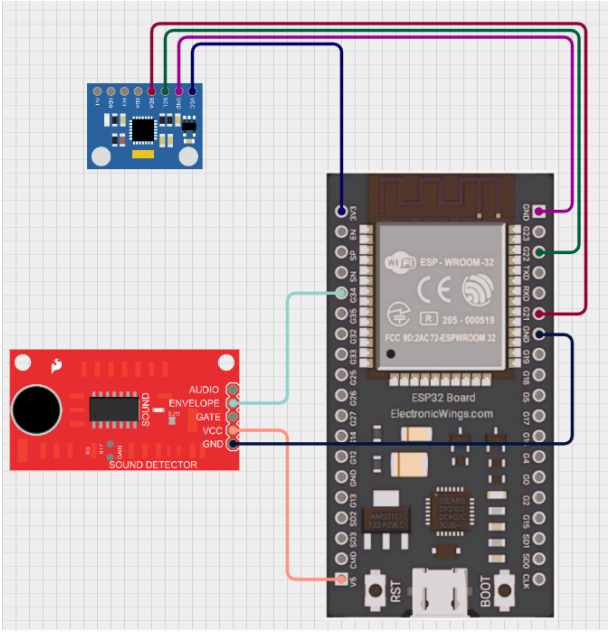


Figura 1: Diagrama de arquitectura

5.2. Conexiones y hardware

- **MPU6050 (conexión I2C):**
  - VCC → 3.3V
  - GND → GND
  - SCL → GPIO22
  - SDA → GPIO21
- **Sound Detector (salida analógica):**
  - VCC → 5V
  - GND → GND
  - ENVELOPE → GPIO34 (entrada analógica)
- **Alimentación:** El ESP32 se alimentó mediante un cable micro-USB conectado a la computadora.

5.3. Sketchs desarrollados

Se desarrollaron dos sketchs en Arduino IDE:

- **Sketch de recolección de muestras:** Capta los datos del acelerómetro y el micrófono a 100Hz e imprime por monitor serie los valores separados por comas en el formato `accX,accY,accZ,micSignal`. Esta salida fue utilizada por la herramienta `edge-impulse-data-forwarder` para subir datos a la plataforma Edge Impulse.
- **Sketch de inferencia:** Acumula las muestras sensadas en un buffer circular, las entrega al modelo embebido ejecutado por el ESP32, y muestra el resultado de

la clasificación por puerto serie. Si el modelo alcanza una probabilidad mayor al umbral configurado (60 %), se informa la clase con mayor confianza en promedio (OFF, NORMAL o ANOMALY). Aquellas muestras que no superen el umbral serán consideradas como Ïnferencia incierta".

6. Recolección de datos

Para entrenar un modelo de clasificación preciso se utilizó como caso práctico un ventilador. Fue necesario recolectar muestras representativas de los tres posibles estados del motor: apagado, funcionando normalmente y funcionando con alguna anomalía.

Se utilizó un sketch en el ESP32 que, a una frecuencia de **100Hz** (frecuencia mínima necesaria para el microfono), capturaba datos simultáneamente del acelerómetro (ejes X, Y, Z) y del micrófono (señal analógica de envolvente), imprimiendo por puerto serie una línea por muestra en el formato:

```
accX,accY,accZ,micSignal
```

Para enviar estas lecturas a Edge Impulse se utilizó la herramienta `edge-impulse-data-forwarder`, que permite leer datos desde el monitor serie y redirigirlos a la plataforma en tiempo real. Esta herramienta detectó automáticamente el dispositivo y permitió desde la interfaz web seleccionar la duración de cada muestra (en este caso, **4 segundos**).

Inicialmente se tomaron 120 muestras (40 por clase), pero al identificar dificultades en la clasificación entre estados NORMAL y ANOMALY, se amplió el dataset a **180 muestras de entrenamiento**: 40 OFF, 60 NORMAL y 80 ANOMALY, procurando incluir mayor variedad de anomalías (desbalanceo, base inestable, choques, cambios de velocidad, etc.).

Adicionalmente se destinaron **45 muestras para validación** (10 OFF, 17 NORMAL y 18 ANOMALY), cumpliendo el criterio recomendado de un 80/20 entre entrenamiento y prueba.

7. Entrenamiento del modelo

Para el entrenamiento del modelo se utilizó la plataforma Edge Impulse, que permite trabajar con datos de series temporales y dispositivos embebidos. Se creó un proyecto nuevo seleccionando la modalidad **Time Series Data**, estableciendo los siguientes parámetros:

- **Window size:** 4000 ms
- **Window increase:** 2000 ms
- **Frecuencia de muestreo:** 100 Hz

Bloques de procesamiento y aprendizaje

- **Bloque de procesamiento:** Se utilizó `Spectral Analysis` para las variables del acelerómetro (`accX`, `accY`, `accZ`) y `Spectrogram` para la señal de micrófono (`micSignal`). En ambos casos se empleó la función *Autotune parameters* para optimizar los parámetros en base a las muestras.

- **Bloque de aprendizaje:** Se seleccionó `Classification` como tipo de modelo. El modelo recibe las características generadas por los bloques anteriores y predice una de las tres clases posibles: `OFF`, `NORMAL` o `ANOMALY`.

Arquitectura de la red neuronal

La arquitectura final estuvo compuesta por:

- Capa de entrada: 90 features
- 1era capa densa: 32 neuronas
- 2da capa densa: 16 neuronas
- 3era capa densa: 8 neuronas
- Capa de salida: 3 neuronas (softmax)

Configuración de entrenamiento

- **Epochs:** 50 ciclos de entrenamiento
- **Learning rate:** 0.005
- **Validation split:** 20 %
- **Entrenamiento en CPU** local
- Activadas las opciones de **Auto-weight classes** y **Profile int8 model**

Resultados

El modelo alcanzó una precisión (*accuracy*) del **94.4 %** con una pérdida (*loss*) del 0.20. La matriz de confusión y las métricas F1-score muestran un buen rendimiento general, con algunas confusiones leves entre los estados NORMAL y ANOMALY, que fueron atenuadas al balancear el dataset y aumentar la diversidad de las anomalías.

8. Entrenamiento del modelo

Para el entrenamiento del modelo se utilizó la plataforma Edge Impulse, que permite trabajar con datos de series temporales y dispositivos embebidos. Se creó un proyecto nuevo seleccionando la modalidad **Time Series Data**, estableciendo los siguientes parámetros:

- **Window size:** 4000 ms
- **Window increase:** 2000 ms
- **Frecuencia de muestreo:** 100 Hz

Bloques de procesamiento y aprendizaje

- **Bloque de procesamiento:** Se utilizó `Spectral Analysis` para las variables del acelerómetro (`accX`, `accY`, `accZ`) y `Spectrogram` para la señal de micrófono (`micSignal`). En ambos casos se empleó la función *Autotune parameters* para optimizar los parámetros en base a las muestras.
- **Bloque de aprendizaje:** Se seleccionó `Classification` como tipo de modelo. El modelo recibe las características generadas por los bloques anteriores y predice una de las tres clases posibles: `OFF`, `NORMAL` o `ANOMALY`.

Arquitectura de la red neuronal

La arquitectura final estuvo compuesta por:

- Capa de entrada: 90 features
- 1era capa densa: 32 neuronas
- 2da capa densa: 16 neuronas
- 3era capa densa: 8 neuronas
- Capa de salida: 3 neuronas (softmax)

Configuración de entrenamiento

- **Epochs:** 50 ciclos de entrenamiento
- **Learning rate:** 0.005
- **Validation split:** 20 %
- **Entrenamiento en CPU** local
- Activadas las opciones de **Auto-weight classes** y **Profile int8 model**

Resultados

El modelo alcanzó una precisión (*accuracy*) del **94.4 %** con una pérdida (*loss*) del 0.20. La matriz de confusión y las métricas F1-score muestran un buen rendimiento general, con algunas confusiones leves entre los estados NORMAL y ANOMALY, que fueron atenuadas al balancear el dataset y aumentar la diversidad de las anomalías.

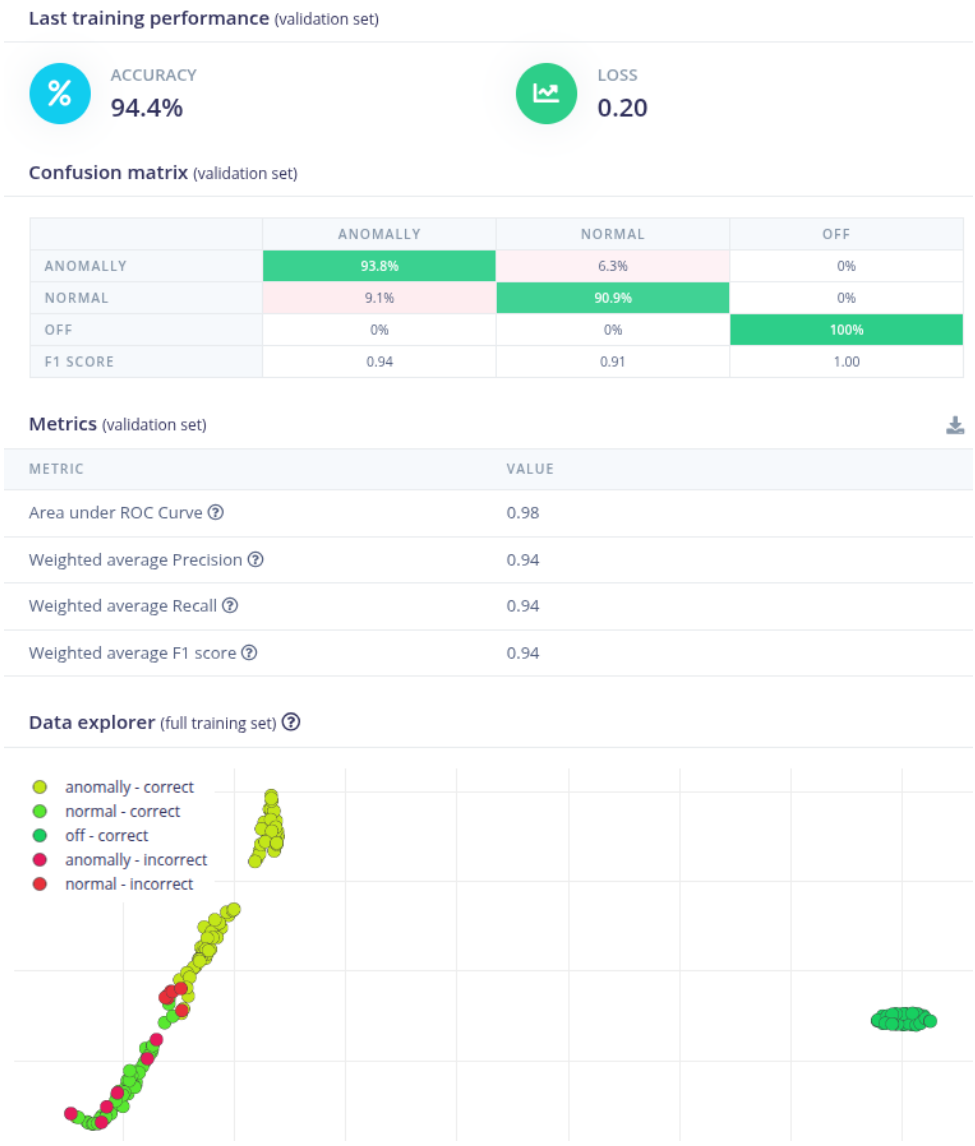


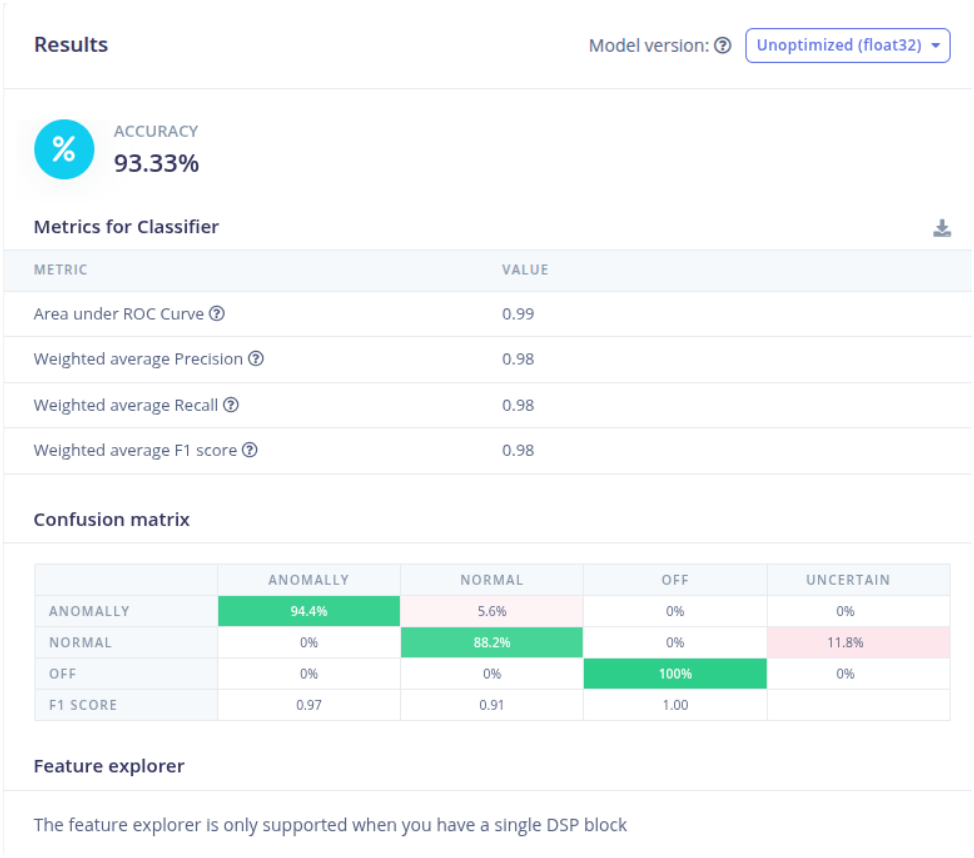
Figura 2: Resultados del modelo de clasificación en Edge Impulse

9. Pruebas y resultados

Una vez entrenado el modelo, se procedió a evaluar su rendimiento tanto en la plataforma Edge Impulse como en pruebas reales sobre el dispositivo.

Validación en Edge Impulse

Se destinaron 45 muestras para prueba (10 OFF, 17 NORMAL y 18 ANOMALY), respetando la proporción del 20 % del dataset total. La validación automática arrojó un **accuracy de 93.33 %** y un **F1-score promedio de 0.98**. El modelo clasificó correctamente todas las muestras etiquetadas como OFF, mientras que mostró una leve confusión entre las clases NORMAL y ANOMALY.



Results — Model version: Unoptimized (float32)

ACCURACY
93.33%

Metrics for Classifier

METRIC	VALUE
Area under ROC Curve	0.99
Weighted average Precision	0.98
Weighted average Recall	0.98
Weighted average F1 score	0.98

Confusion matrix

	ANOMALLY	NORMAL	OFF	UNCERTAIN
ANOMALLY	94.4%	5.6%	0%	0%
NORMAL	0%	88.2%	0%	11.8%
OFF	0%	0%	100%	0%
F1 SCORE	0.97	0.91	1.00	

Feature explorer

The feature explorer is only supported when you have a single DSP block

Figura 3: Resultado de testing

Pruebas en dispositivo real

El modelo fue cargado en el ESP32 mediante la exportación como librería Arduino proporcionada por Edge Impulse. La plataforma ofrece dos tipos de modelos para exportar, **Quantized (int8)** y **Unoptimized (float32)**, en este caso se optó por usar Quantized debido a que este último posee menor latencia y menor consumo de memoria.

Se utilizó un segundo sketch que, cada 10ms, leía los valores de los sensores y acumulaba 400 muestras para realizar una inferencia.

La inferencia se ejecuta cada **4 segundos** aproximadamente, y si la probabilidad de la clase ganadora supera el umbral de confianza (0.6), se muestra por el monitor serie. Si no, se indica como *inferencia incierta*".

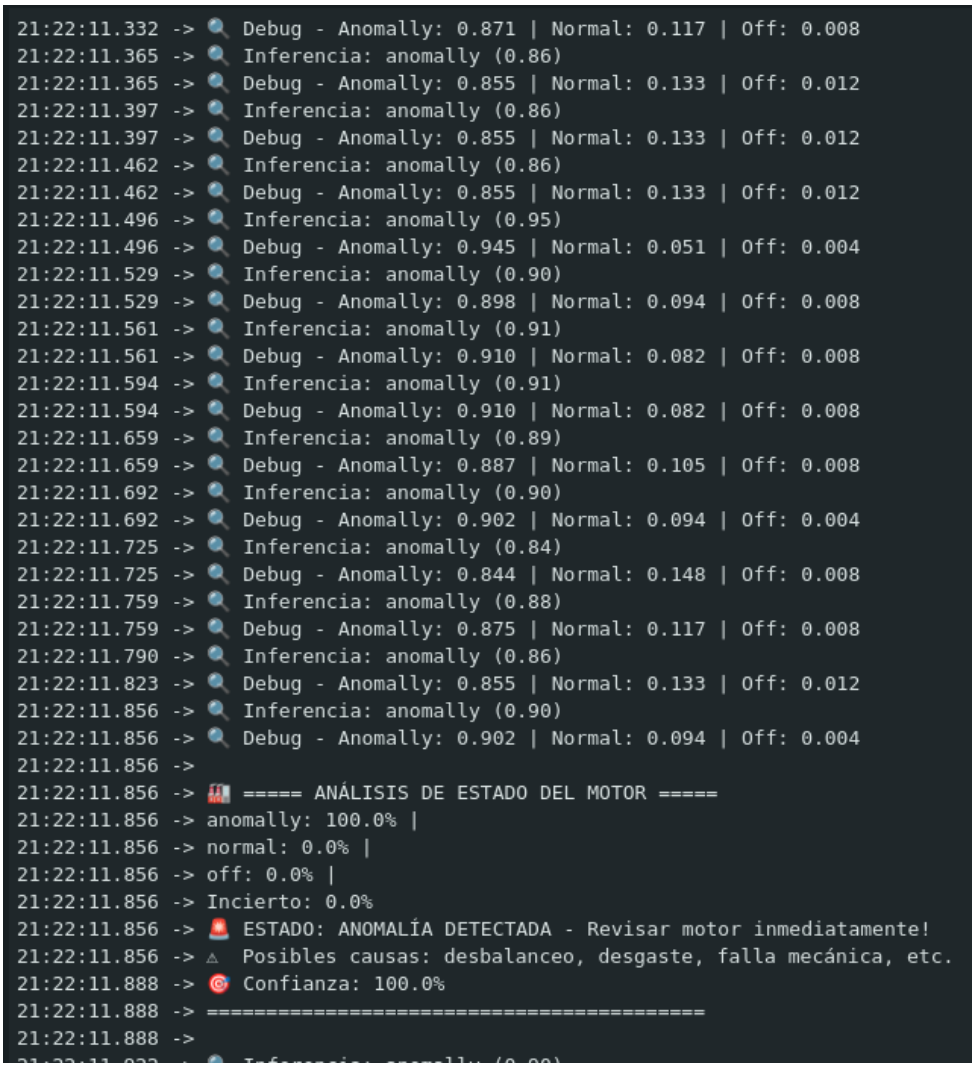


Figura 4: Inferencia del estado ANOMALY en tiempo real

En todas las pruebas realizadas con el ventilador, el sistema logró identificar correctamente los tres estados: apagado, funcionamiento normal y funcionamiento anómalo. Esto validá el buen desempeño del modelo en condiciones reales y refuerza su utilidad en escenarios sin conectividad externa.

10. Problemas y soluciones

Durante el desarrollo del proyecto surgieron distintos desafíos que debieron abordarse mediante pruebas, ajustes e iteraciones sucesivas. A continuación, se detallan los principales problemas identificados y las soluciones implementadas:

- **Señales insuficientes con el cooler de PC:** Inicialmente se utilizó un cooler de PC como fuente de prueba para el sistema. Sin embargo, se detectáron dificultades para distinguir entre los estados OFF y NORMAL debido a la baja generación de vibraciones y sonido. ***Solución:*** Se reemplazó por un ventilador convencional, cuyas señales eran más claras y marcadas.
- **Confusión entre clases NORMAL y ANOMALY:** Al realizar las primeras pruebas, el modelo presentaba dificultades para diferenciar entre el funcionamiento

normal y las anomalías. ***Solución:*** Se amplió el dataset, aumentando la cantidad de muestras NORMAL y ANOMALY, y se agregó mayor diversidad de anomalías (desbalanceo, base inestable, cambios de velocidad, golpes, etc.).

- **Dataset inicial limitado:** El conjunto de datos original (120 muestras) no era suficiente para un modelo robusto. ***Solución:*** Se aumentó a 180 muestras de entrenamiento y 45 de prueba.
- **Anomalías poco variadas:** Las primeras anomalías simuladas eran repetitivas (papeles y golpes leves), lo que limitaba la generalización del modelo. ***Solución:*** Se incorporó variedad en la simulación de fallas y combinaciones de anomalías.

Estas decisiones permitieron mejorar la calidad del entrenamiento, el rendimiento del modelo y su aplicabilidad a situaciones reales.

11. Conclusión

El desarrollo de este proyecto permitió explorar de manera integral una solución Edge para la detección de anomalías en motores utilizando machine learning y sensores accesibles. A partir de la integración del ESP32 con el acelerómetro MPU6050 y el sensor de sonido Sound Detector, fue posible capturar datos relevantes que reflejan el estado de funcionamiento del motor.

Gracias a la plataforma Edge Impulse se logró entrenar un modelo con buena precisión, desplegarlo en el dispositivo y validar su funcionamiento tanto en pruebas simuladas como reales. El modelo alcanzó una exactitud del 94.4 % y demostró un comportamiento robusto frente a los distintos estados clasificados.

Se comprobó que la selección del hardware y la variedad en la recolección de datos son factores determinantes para el éxito del modelo. Además, el hecho de que la inferencia ocurra localmente, sin depender de internet ni de procesamiento externo, refuerza la utilidad de esta solución en contextos rurales o de bajo presupuesto.

En casos como el cooler (o incluso otro dispositivo de prueba), sería correcto acompañar al modelo con otro tipo de sensor, como por ejemplo, un sensor de corriente.

Este trabajo demostró que es viable construir sistemas inteligentes de monitoreo a partir de plataformas de bajo costo, siempre que se realice un proceso iterativo de recolección, prueba y ajuste. Los resultados obtenidos validan la propuesta y abren la puerta a mejoras futuras.

12. Trabajos futuros

Si bien el sistema desarrollado cumple con su objetivo de detectar el estado de un motor de forma local y con buena precisión, existen diversas oportunidades para ampliar sus capacidades y mejorar su aplicabilidad en escenarios reales. A continuación se detallan algunas propuestas:

- **Persistencia de datos:** Registrar las inferencias realizadas en una base de datos local o remota (por ejemplo, InfluxDB) para su posterior análisis y visualización.

- **Visualización en tiempo real:** Integrar el sistema con herramientas como Grafana para mostrar dashboards con el historial de estados y generar alertas automáticas ante la detección de anomalías.
- **Interfaces de comunicación:** Exponer los resultados a través de un servidor web, un bot de Telegram, MQTT u otros medios que permitan notificar al usuario en tiempo real.
- **Clasificación más detallada:** Ampliar el modelo para que pueda distinguir entre distintos tipos de anomalías (desbalanceo, obstrucción, sonido anormal, vibración excesiva, etc.).
- **Actualización continua del modelo:** Incorporar mecanismos de aprendizaje incremental para actualizar el modelo en base a nuevas muestras recolectadas durante su uso.
- **Adaptación a otros entornos:** Aplicar esta solución a otros dispositivos o máquinas, como bombas, extractores o herramientas eléctricas, posiblemente con sensores alternativos (temperatura, consumo eléctrico, presión, etc.).
- **Optimizaciones técnicas:** Mejorar la eficiencia energética del sistema, reducir el uso de memoria y optimizar la latencia de inferencia para uso en dispositivos portátiles o con batería.

Estas mejoras permitirían llevar la solución hacia aplicaciones industriales reales, integrarla en sistemas de monitoreo más complejos y extender su alcance a diferentes tipos de equipamiento y condiciones de operación.

13. Bibliografía y enlaces

- Repositorio del proyecto en GitHub
- Proyecto público de Edge Impulse
- Documentación MPU-6050
- Documentación SparkFun Sound Detector
- Instalación de CLI de Edge Impulse
- Uso de edge-impulse-data-forwarder
- Teorías de la matería Internet de las Cosas 2025.