Universidad Católica de Santa María

Facultad de Ciencias e Ingenierías Físicas y Formales

Escuela Profesional de Ingeniería Electrónica

Diseño de un sistema predictivo con arquitectura embebida distribuida para el monitoreo del estado operativo y análisis de degradación en motores eléctricos mediante sincronización de datos vía MQTT.

Tesis presentada por:

Rivera Linares, Reymer Alonso

ORCID: 0009-0008-9517-0550

para optar el Título Profesional de Ingeniero Electrónico con especialidad en Automatización y Control

Asesor (a):

Dr. Copa Pineda, Juan Carlos

ORCID: 0000-0002-8884-2039

Arequipa - Perú

2026

UCSM-ERP

UNIVERSIDAD CATÓLICA DE SANTA MARÍA

INGENIERIA ELECTRONICA

CON ESPECIALIDAD EN AUTOMATIZACIÓN Y CONTROL

TITULACIÓN CON TESIS

DICTAMEN APROBACIÓN DE BORRADOR

Arequipa, 25 de Marzo del 2026

Dictamen: 016582-C-EPIE-2026

Visto el borrador del expediente 016582, presentado por:

2007603081 - **RIVERA LINARES REYMER ALONSO**

Titulado:

DISEÑO DE UN SISTEMA PREDICTIVO CON ARQUITECTURA EMBEBIDA DISTRIBUIDA PARA EL MONITOREO DEL ESTADO OPERATIVO Y ANÁLISIS DE DEGRADACIÓN EN MOTORES ELÉCTRICOS MEDIANTE SINCRONIZACIÓN DE DATOS VÍA MQTT.

Nuestro dictamen es:

APROBADO

Titulo Profesional/Titulo de Segunda Especialidad/Grado Académico a optar:

INGENIERO ELECTRONICO CON ESPECIALIDAD EN AUTOMATIZACIÓN Y CONTROL

29364669 - **QUISPE YAUYO JUAN MEDARDO**
DICTAMINADOR

29410027 - **SULLA TORRES RAUL RICARDO**
DICTAMINADOR

29720651 - **ZEGARRA GAGO HENRY CHRISTIAN**
DICTAMINADOR

DISEÑO DE UN SISTEMA PREDICTIVO CON ARQUITECTURA EMBEBIDA DISTRIBUIDA PARA EL MONITOREO DEL ESTADO OPERATIVO Y ANÁLISIS DE DEGRADACIÓN EN MOTORES ELÉCTRICOS MEDIANTE SINCRONIZACIÓN DE DATOS VÍA MQTT.

INFORME DE ORIGINALIDAD

11%	9%	2%	4%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	Submitted to Universidad Católica de Santa María Trabajo del estudiante	1%
2	dspace.ups.edu.ec Fuente de Internet	1%
3	repositorio.unjfsc.edu.pe Fuente de Internet	1%
4	www.coursehero.com Fuente de Internet	<1%
5	repositorio.continental.edu.pe Fuente de Internet	<1%
6	repositorio.uncp.edu.pe Fuente de Internet	<1%
7	repositorio.uancv.edu.pe Fuente de Internet	<1%

Dedicatoria

Dedico esta tesis a Dios, a mi familia, a mi esposa y a mi niña, quienes me apoyaron y confiaron en mi durante todo este proceso.

Agradecimientos

Agradezco a mi universidad alma mater y a sus autoridades por brindarme las herramientas necesarias para mi formación profesional, así como a los Ingenieros con quienes tuve la oportunidad de compartir y que contribuyeron en este proceso académico.

RESUMEN

La investigación desarrolla un sistema predictivo con arquitectura embebida distribuida, orientado al monitoreo del estado operativo y análisis de degradación en motores eléctricos, empleando sincronización de datos mediante el protocolo MQTT. El objetivo principal fue diseñar una solución capaz de anticipar fallas incipientes y optimizar la gestión del mantenimiento industrial. El sistema se estructuró en tres nodos edge que permiten la adquisición y procesamiento local de datos de vibración, temperatura y corriente, garantizando una comunicación eficiente con latencias inferiores a 50 ms, throughput promedio de 1 msg/s y disponibilidad del 100 % en los nodos principales. El uso del protocolo MQTT con niveles de servicio QoS 0-2 aseguró la sincronización y transmisión confiable de datos, con pérdida nula de mensajes y una deriva temporal menor a 2.2 ms entre nodos, validando su estabilidad en entornos de conectividad variable. Para el análisis predictivo se integraron los modelos XGBoost y Weibull-AFT. El primero permitió clasificar la tendencia de falla bajo tres escenarios (base, conservador y estresado), alcanzando una AUC-ROC de 0.988 y alta precisión en las curvas de calibración y precisión-recuperación. El segundo modelo estimó una vida útil remanente (RUL) de 10,932 horas, equivalente a 455 días de operación continua, con un Brier Score de 0.1922 y C-index de 0.1343, demostrando una calibración adecuada y coherencia entre las predicciones y los eventos reales de falla. Las curvas de supervivencia mostraron una disminución progresiva del RUL en un 22.7 % a lo largo de 24 meses. Los resultados confirman que la integración de procesamiento distribuido, comunicación IoT y modelos de predicción híbridos mejora la capacidad de diagnóstico en tiempo real, reduce el riesgo de fallas y permite planificar intervenciones preventivas con mayor exactitud. El sistema propuesto demostró su viabilidad técnica, precisión superior al 98 % y reducción de más del 20 % en las paradas no planificadas, consolidándose como una herramienta eficaz para el mantenimiento predictivo en entornos industriales eléctricos.

Palabras Clave: Arquitectura embebida, Mantenimiento predictivo, Motores eléctricos.

ABSTRACT

The research develops a predictive system with distributed embedded architecture, aimed at monitoring the operating status and analyzing degradation in electric motors, using data synchronization via the MQTT protocol. The main objective was to design a solution capable of anticipating incipient failures and optimizing industrial maintenance management. The system was structured into three edge nodes that enable the local acquisition and processing of vibration, temperature, and current data, ensuring efficient communication with latencies of less than 50 ms, average throughput of 1 msg/s, and 100% availability at the main nodes. The use of the MQTT protocol with QoS 0-2 service levels ensured reliable data synchronization and transmission, with zero message loss and a time drift of less than 2.2 ms between nodes, validating its stability in environments with variable connectivity. The XGBoost and Weibull-AFT models were integrated for predictive analysis. The former allowed the failure trend to be classified under three scenarios (base, conservative, and stressed), achieving an AUC-ROC of 0.988 and high accuracy in the calibration and accuracy-recovery curves. The second model estimated a remaining useful life (RUL) of 10,932 hours, equivalent to 455 days of continuous operation, with a Brier Score of 0.1922 and C-index of 0.1343, demonstrating adequate calibration and consistency between predictions and actual failure events. The survival curves showed a progressive decrease in RUL of 22.7% over 24 months. The results confirm that the integration of distributed processing, IoT communication, and hybrid prediction models improves real-time diagnostic capabilities, reduces the risk of failure, and allows for more accurate planning of preventive interventions. The proposed system demonstrated its technical feasibility, accuracy greater than 98%, and reduction of more than 20% in unplanned outages, establishing itself as an effective tool for predictive maintenance in industrial electrical environments.

Keywords: Embedded architecture, Predictive maintenance, Electric motors.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ANEXOS

INTRODUCCIÓN

En la industria, los motores eléctricos son esenciales para procesos como bombeo, transporte y producción continua, donde su desempeño influye directamente en la eficiencia operativa. La necesidad de maximizar su disponibilidad ha impulsado la adopción de tecnologías que permitan supervisar y anticipar fallas, reduciendo costos y evitando paradas no programadas. Las arquitecturas embebidas distribuidas con nodos de procesamiento local (edge computing) representan una solución eficaz para el monitoreo en tiempo real. Estos nodos permiten capturar, procesar y filtrar datos de variables como vibración, temperatura y corriente antes de transmitirlos, reduciendo la latencia y optimizando el uso de la red. El protocolo MQTT, por su estructura ligera y esquema publish/subscribe, es adecuado para la sincronización de datos en entornos industriales IoT, ofreciendo eficiencia en el uso de ancho de banda y adaptabilidad a redes con diferentes calidades de servicio. La incorporación de algoritmos de machine learning, como XGBoost y Weibull-AFT, facilita la detección de tendencias de falla y la estimación de la vida útil remanente (RUL), permitiendo una planificación de mantenimiento más precisa. Esta investigación se centra en el diseño y simulación de un sistema predictivo que combine estas tecnologías, con validación en entornos controlados para evaluar su precisión y aplicabilidad industrial.

CAPÍTULO I

PLANTEAMIENTO DE ESTUDIO

1.1 Identificación del Problema

En el ámbito industrial, los motores eléctricos son componentes críticos para la operación de líneas de producción, sistemas de bombeo, transporte de materiales y otros procesos. Su correcta operación impacta directamente en la continuidad productiva, y cualquier interrupción no planificada puede generar pérdidas económicas significativas. De acuerdo con ABB (2023), el costo por hora de inactividad no planificada puede alcanzar los USD 125 000 y representar alrededor del 11% de los ingresos en fabricantes de gran escala. Esto ha incrementado la presión para implementar estrategias de mantenimiento predictivo y sistemas de monitoreo continuo del estado operativo.

Actualmente, gran parte de los esquemas de monitoreo en uso se basan en arquitecturas centralizadas o en campañas de medición periódica. Estos enfoques presentan limitaciones en la capacidad de correlacionar múltiples variables (vibración, temperatura, corriente, entre otras) en tiempo cercano al real, lo que retrasa la identificación de patrones de degradación. Según Karakolev y Dimitrov (2018), los rodamientos representan aproximadamente el 50 % de las fallas mecánicas en motores eléctricos, lo que evidencia la necesidad de un análisis más granular para identificar y anticipar estos fallos.

En términos de normativa, el diagnóstico de severidad de vibraciones debe alinearse con estándares internacionales como la ISO 10816/20816, que establecen umbrales de referencia para distintos tipos y tamaños de maquinaria. Por ejemplo, en bombas y motores, valores globales superiores a 2.8 mm/s RMS suelen indicar condiciones de operación restringida o inaceptable (Sánchez et al., 2025). Para garantizar comparaciones confiables con estos límites, es imprescindible contar con adquisición de datos precisa, sincronizada y resistente a interrupciones de comunicación. La implementación de arquitecturas embebidas distribuidas, con nodos de procesamiento local (“edge”), ofrece ventajas frente a los sistemas centralizados: reduce la latencia, disminuye la carga de transmisión de datos y permite mantener capacidad de diagnóstico aun en entornos con conectividad inestable. Estos nodos pueden filtrar y procesar información antes de enviarla, optimizando el uso de recursos de red y mejorando la velocidad de respuesta. En base a lo mencionado anteriormente, el protocolo MQTT se presenta como una alternativa eficiente para la sincronización de datos entre nodos

y un servidor central o “broker”. Sus características cabeceras ligeras (~2 bytes), soporte de calidad de servicio (QoS), retención de mensajes y arquitectura publish/subscribe lo hacen más adecuado que HTTP para aplicaciones IoT (EMQX, 2024). Sin embargo, en muchas instalaciones industriales aún no se cuenta con una solución que integre procesamiento distribuido en el borde, comunicación mediante MQTT y modelos predictivos capaces de estimar la degradación y riesgo de falla, respetando estándares internacionales de severidad de vibraciones.

1.2 Descripción del problema

En la operación industrial, los motores eléctricos suelen presentar fallas mecánicas y eléctricas que provocan paradas no planificadas y aumentan los costos. Los sistemas de monitoreo actuales, mayormente centralizados o con mediciones periódicas, no detectan a tiempo la degradación de componentes como rodamientos. Además, la falta de sincronización de datos y análisis inmediato dificulta correlacionar variables como vibración, temperatura y corriente. La ausencia de soluciones integradas que combinen procesamiento distribuido en el borde, transmisión de datos eficiente mediante MQTT y modelos predictivos limita la capacidad de diagnóstico continuo. Esto impide anticipar fallas y reducir el impacto económico derivado del tiempo de inactividad no programado en las operaciones industriales.

1.3 Objetivos

1.3.1 Objetivo General

Diseñar un sistema predictivo con arquitectura embebida distribuida para el monitoreo del estado operativo y análisis de degradación en motores eléctricos, utilizando sincronización de datos vía MQTT.

1.3.2 Objetivos Específicos

a) Identificar las variables de operación que influyen en la degradación de motores.
b) Desarrollar la arquitectura embebida distribuida para adquisición y procesamiento local de datos.
c) Implementar la transmisión y sincronización de datos mediante MQTT.

d) Integrar dos modelos predictivos: XGBoost para clasificar y predecir la tendencia de falla, y Weibull-AFT para estimar riesgo y vida útil remanente.
e) Validar el sistema predictivo propuesto mediante métricas de desempeño.

1.4 Variables

Variable Independiente: Arquitectura embebida distribuida con sincronización de datos vía MQTT

Indicadores:

- Variables de operación (vibración, temperatura y corriente)
- Nodos de procesamiento local “edge”,
- Tasa de pérdida de mensajes (%)
- Throughput (msg/s)
- Frecuencia de muestreo (Hz)
- Latencia de transmisión (ms)
- Ancho de banda utilizado (kbps)
- Disponibilidad del sistema (%)

*

Variable Dependiente: Estado operativo y análisis de degradación en motores eléctricos

Indicadores:

- Métricas de desempeño (Accuracy, F1, ROC-AUC) con XGBoost.
- Error de regresión del índice de degradación (RMSE/MAE).
- Tiempo de alerta previa (h/días).
- Desempeño de supervivencia (Weibull-AFT): C-index y Brier score.
- Error en RUL (MAPE del RUL estimado).
- Tasa de falsas alarmas y alarmas omitidas (por mes).
- Variación de MTBF y MTTR
- Reducción de paradas no planificadas (%).

1.5 Hipótesis

1.5.1 Hipótesis General

Un sistema predictivo con arquitectura embebida distribuida y sincronización de datos vía MQTT mejora la precisión y oportunidad en el monitoreo del estado operativo y análisis de degradación de motores eléctricos, reduciendo el riesgo de fallas y paradas no planificadas.

1.5.2 Hipótesis Específicas

a) La identificación de variables de operación aumenta la exactitud en la detección de degradación en motores eléctricos.
b) Una arquitectura embebida distribuida optimiza la adquisición y el procesamiento local de datos, reduciendo la latencia del sistema.
c) La transmisión y sincronización de datos mediante MQTT disminuye la pérdida de información y mejora la confiabilidad de la comunicación.
d) El uso de modelos predictivos XGBoost y Weibull-AFT incrementa la precisión en la estimación de la tendencia de falla y la vida útil remanente.
e) La validación con métricas de desempeño evidencia mejoras en la predicción y monitoreo frente a métodos convencionales.

1.6 Alcances

La presente investigación se desarrollará a nivel de diseño y simulación, sin contemplar la implementación física del sistema propuesto. El alcance incluye el modelado de una arquitectura embebida distribuida para el monitoreo del estado operativo y análisis de degradación en motores eléctricos, incorporando la sincronización de datos mediante el protocolo MQTT.

El trabajo abarcará:

- Identificación de variables de operación relevantes (vibración, temperatura y corriente).
- Desarrollo del esquema de adquisición y procesamiento local de datos en nodos distribuidos, con evaluación de latencia, disponibilidad y uso de ancho de banda.

- Integración de los modelos predictivos XGBoost y Weibull-AFT para la clasificación de tendencias de falla y la estimación de la vida útil remanente (RUL).
- Simulación y validación del sistema mediante métricas de desempeño (Accuracy, F1, ROC-AUC, C-index, Brier score), comparando con métodos de monitoreo convencionales.
- Análisis de resultados enfocado en la reducción de pérdidas de información, mejora en la confiabilidad de comunicación y optimización de la detección temprana de degradación.

El alcance no considera la adquisición de hardware definitivo, pruebas en campo ni implementación en una planta industrial real; las validaciones se limitarán a entornos de simulación controlada.

CAPÍTULO II

MARCO TEÓRICO

2.1 Antecedentes de la investigación

2.1.1 Antecedentes Nacionales

Torres y Escobar (2024) en su tesis *"Diseño e implementación de un sistema predictor de fallas con redes neuronales para motores trifásicos en el sector industrial"*, tuvieron como objetivo diseñar e implementar un sistema automatizado de predicción de fallas en motores eléctricos trifásicos, empleando redes neuronales artificiales para mejorar la confiabilidad del diagnóstico y reducir paradas no programadas. La metodología utilizada fue de enfoque cuantitativo, con alcance descriptivo, siguiendo la norma VDI 2221 para el desarrollo de sistemas mecatrónicos. El sistema se implementó en un banco de pruebas con sensores de temperatura, vibración, corriente y voltaje, conectados a un PLC y monitoreados en tiempo real mediante HMI. Los datos adquiridos se procesaron y almacenaron para su análisis con un algoritmo de red neuronal. Los resultados demostraron que el sistema alcanzó un 99 % de eficiencia en la predicción de fallas, validándose su precisión frente a mediciones de campo, lo que confirma su viabilidad técnica y económica en entornos industriales.

Carpio (2025) en su tesis *"Diseño de un sistema de mantenimiento predictivo con machine learning aplicado a vibraciones de motores eléctricos"*, tuvo como objetivo diseñar un sistema de mantenimiento predictivo para motores eléctricos, basado en el análisis de vibraciones y algoritmos de machine learning, con el fin de detectar fallas mecánicas de forma anticipada y reducir paradas no programadas. La metodología contempló el diseño de una arquitectura modular con sensores triaxiales, adquisición y preprocesamiento de señales, extracción de características (RMS, curtosis, skewness, espectro de frecuencia) y clasificación automática mediante el algoritmo Random Forest. Se desarrollaron etapas de recolección de datos reales, selección de modelos, entrenamiento, validación y visualización con alertas automáticas. Los resultados evidenciaron que el sistema propuesto permite identificar patrones de fallas como desbalanceo y defectos en rodamientos, emitiendo diagnósticos automáticos confiables, mejorando la planificación del mantenimiento, optimizando recursos y aumentando la disponibilidad de los motores.

Cordova (2025) en su investigación *"Implementación de plan de mantenimiento predictivo en motores del sistema de enfriamiento de variadores de velocidad de bombas Geho"*, tuvo como objetivo implementar un plan de mantenimiento predictivo en los motores del sistema de enfriamiento de los variadores de velocidad de las bombas GEHO en Minera Chinalco, con el fin de optimizar disponibilidad, confiabilidad y vida útil, reduciendo costos y tiempos de inactividad. La metodología consideró análisis de criticidad, instalación de sensores y técnicas como análisis de vibraciones (límites según ISO 2372 entre 0,18 y 28 mm/s), termografía (temperaturas máximas de 120 °C a 240 °C según clase de aislamiento), inspecciones boroscópicas y análisis de aceites. Los resultados incluyeron la reducción de fallas inesperadas, aumento del caudal operativo (hasta 28 L/s en condiciones actuales), prolongación de la vida útil de los motores y optimización de recursos, además de mejoras en seguridad al evitar sobrecalentamientos y sobrepresiones, como las limitadas a 8 910 kPa en el sistema.

2.1.2 Antecedentes Internacionales

Gómez (2025) en su tesis *"Sistema de adquisición y monitoreo en tiempo real para la detección temprana de fallas en motores mediante análisis de señales de vibración y temperatura bajo un ambiente (IoT)"*, tuvo como objetivo diseñar, construir e integrar un sistema embebido de adquisición y monitoreo en tiempo real para detectar fallas tempranas en motores asíncronos, mediante el análisis de vibraciones y temperatura en un entorno IoT, validado en un motor industrial de 55 kW. La metodología consistió en un diseño experimental con sensores de vibración y temperatura conectados a un prototipo IoT que enviaba datos cada 15 segundos a la nube (ThingSpeak), procesados y graficados en Python para su análisis. Se realizaron pruebas en arranque, desaceleración y operación continua. Los resultados evidenciaron que el sistema detecta eficazmente anomalías, como picos de vibración superiores a 24 mm/s y temperaturas de hasta 49,5 °C, generando alertas automáticas. Se concluyó que la solución es viable para mantenimiento predictivo, optimizando la intervención antes de fallas críticas.

Castillo (2022) en su tesis *"Sistema de detección predictiva de fallas de un motor de baja tensión mediante protocolo MQTT y aplicación IoT"*, tuvo como objetivo desarrollar un sistema de detección predictiva de fallas en un motor de baja tensión mediante protocolo MQTT y aplicación IoT, para monitorear variables como corriente, temperatura y vibración,

optimizando el mantenimiento en la planta cementera UCEM. La metodología empleada combinó observación directa en campo, métodos cualitativo y cuantitativo para el análisis de información, métodos analítico y deductivo para el tratamiento de datos, y la metodología Agile para el desarrollo y validación del prototipo. Los resultados demostraron que el sistema transmitió datos de manera fiable vía red WiFi, integrando sensores y plataformas como NodeMCU, Raspberry Pi, Node-RED e InfluxDB. Las pruebas evidenciaron un error de medición bajo (corriente: -0,13 A; temperatura: 0,97 °C; vibración dentro de rango), validando la exactitud y funcionalidad del prototipo para generar alertas predictivas y facilitar la toma de decisiones de mantenimiento.

Farez y Auqui (2024) en su tesis *"Diseño e implementación de un sistema de monitoreo IoT de Temperatura y Vibración para protección de Motores Eléctricos"*, tuvieron como objetivo implementar un sistema IoT para monitorear temperatura y vibración en motores eléctricos, reduciendo mantenimientos correctivos y mejorando su vida útil. La metodología incluyó el diseño de un prototipo con sensores SICK MPB10 conectados a un Gateway SIG200 y un PLC Siemens S7-1200, integrados mediante Node-RED para capturar, procesar y visualizar datos en tiempo real. Se configuraron umbrales de alerta y se realizaron pruebas tanto en vacío como con carga acoplada, registrando valores de vibración, temperatura, corriente y velocidad en distintas frecuencias. Los resultados demostraron que el sistema detecta incrementos anómalos de vibración y temperatura, permitiendo identificar resonancias y posibles fallas mecánicas antes de que se produzcan daños graves. Esto valida su eficacia para mantenimiento predictivo, optimiza la operación y disminuye costos asociados a paradas imprevistas.

2.2 Bases Teóricas

2.2.1 Sistema Predictivo

Un sistema predictivo es una herramienta tecnológica que emplea información pasada, algoritmos y modelos estadísticos para prever eventos, comportamientos o tendencias. Estos sistemas examinan patrones y relaciones en grandes cantidades de datos, lo que permite a las organizaciones tomar decisiones basadas en información y con un enfoque estratégico. Son importantes en sectores como finanzas, salud, marketing y manufactura, ya que ayudan a mejorar los procesos, reducir costos y aumentar la eficiencia operativa. Al

anticipar situaciones futuras, las empresas pueden ajustarse de manera anticipada a los cambios en el mercado y a las demandas de los clientes (Pintado, 2008, p.5).

El diseño de un sistema predictivo utilizando algoritmos de bosques aleatorios para calcular el tiempo de mantenimiento en motores trifásicos de corriente alterna se basa en la capacidad de estos motores para generar un campo magnético rotatorio que permite su operación (Pintado, 2008, p.7). Los motores trifásicos permiten predecir su tiempo de mantenimiento es importante para evitar fallos imprevistos. Los bosques aleatorios, una técnica de aprendizaje supervisado, facilitan el análisis de grandes volúmenes de datos históricos sobre el funcionamiento y fallos de los motores. Estos algoritmos crean varios árboles de decisión que predicen patrones de desgaste, identificando variables como el tiempo de uso, la temperatura o la carga, para estimar de manera precisa cuándo será necesario realizar el mantenimiento (Ayala, 2017, p.36).

El sistema predictivo desarrollado con bosques aleatorios trabaja identificando patrones recurrentes de fallos en motores trifásicos, lo que facilita anticipar cuándo es probable que un motor requiera intervención. Para llevar a cabo este sistema, es necesario disponer de una base de datos amplia que registre eventos de mantenimiento anteriores, fallos, tiempos de funcionamiento y condiciones de carga. Con esta información, los bosques aleatorios generan varios árboles de decisión que dividen las variables relevantes, formando un modelo predictivo sólido. Este modelo ayuda a mejorar la eficiencia operativa y disminuir costos, al prever el momento adecuado para realizar mantenimiento preventivo en los motores (Plata y Jurado, 2020, p.45).

Un sistema predictivo es una herramienta que utiliza información, algoritmos y modelos estadísticos para prever eventos o comportamientos futuros. Estos sistemas son clave en sectores como educación, salud, finanzas y manufactura, ya que ayudan a reconocer patrones, anticipar tendencias y mejorar procesos (Plata y Jurado, 2020, p.46).

Un sistema predictivo es un conjunto de herramientas que utiliza técnicas matemáticas complejas para prever eventos futuros a partir de datos pasados. El objetivo principal de un sistema predictivo es encontrar una relación matemática entre una variable dependiente, que representa el resultado o objetivo a predecir, y varias variables independientes o predictoras. Este proceso consiste en estimar los valores futuros de estas

variables independientes, aplicarlos en la ecuación matemática previamente determinada y, de esta forma, generar previsiones sobre el comportamiento futuro de la variable dependiente. La exactitud de estas previsiones depende en gran medida de la calidad de los datos utilizados (Bernaola y Varillas, 2022, p. 76).

Un sistema predictivo es una herramienta útil que ayuda a las organizaciones a tomar decisiones más fundamentadas y estratégicas al prever eventos futuros. Su aplicación se ha extendido a diversas industrias, como la gestión de inventarios, el mantenimiento preventivo, la detección de fraudes y la mejora de la producción. No obstante, el éxito de la implementación de estos sistemas depende en gran medida de la calidad de los datos y de una evaluación y ajuste constante del modelo predictivo (Garcia, 2021). Un sistema predictivo es un conjunto de métodos matemáticos cuyo objetivo es establecer una relación cuantitativa entre una variable dependiente o resultado esperado y diversas variables independientes. Su propósito es prever los valores futuros de esas variables, integrarlos en la relación matemática encontrada y así anticipar posibles valores futuros del resultado o variable dependiente. Dado que estas relaciones rara vez son exactas en la práctica, es importante proporcionar una medida de incertidumbre o confiabilidad asociada a las predicciones realizadas para garantizar una mayor precisión en los resultados (IBM, 2012).

Un sistema predictivo es un conjunto de métodos estadísticos y matemáticos utilizados para identificar la relación entre una variable dependiente, también llamada objetivo o respuesta, y varias variables independientes o factores de predicción. Su propósito es anticipar los valores futuros de estos factores y, mediante esta relación matemática, estimar los valores futuros de la variable dependiente. Debido a que en la práctica estas correlaciones rara vez son completamente exactas, es importante incluir una medida de incertidumbre en las predicciones, lo que permite evaluar la confiabilidad y exactitud de los resultados obtenidos (López et al., 2015, p.12).

Los elementos principales de un sistema predictivo incluyen la obtención de datos, el procesamiento inicial, la elección de modelos, el entrenamiento y la evaluación. La obtención de datos consiste en recopilar información relevante de diversas fuentes, como bases de datos internas, sensores o redes sociales. El procesamiento inicial asegura la calidad de los datos a través de su depuración, normalización y transformación. La elección de modelos consiste en seleccionar algoritmos adecuados, como regresión, árboles de decisión o redes neuronales. El

entrenamiento utiliza conjuntos de datos para ajustar el modelo, mientras que la evaluación verifica su exactitud y capacidad para generalizar, asegurando resultados confiables (Pintado, 2008, p.55).

Para identificar los elementos que afectan la predicción, se pueden agrupar en tres categorías: los que tienen baja probabilidad de influir en el resultado, los que con mayor certeza podrían impactar el resultado y deben ser incluidos en el sistema, y un tercer grupo que se encuentra en un punto intermedio. Estos últimos pueden o no afectar el resultado final, por lo que es importante utilizar diferentes métodos para evaluar si es adecuado incluirlos en el sistema. Esta clasificación ayuda a analizar los factores importantes en la predicción (López et al., 2015, p.32).

El uso de un sistema predictivo en el mantenimiento de motores trifásicos de corriente alterna es una herramienta importante para la industria, ya que permite a las empresas adelantarse a posibles fallos y realizar intervenciones preventivas. Con el apoyo de los algoritmos de bosques aleatorios, el sistema predictivo puede actualizarse continuamente con nuevos datos, mejorando su exactitud y capacidad de adaptación. Esto garantiza que las operaciones industriales sigan siendo eficientes, reduciendo el tiempo de inactividad y los costos asociados a reparaciones inesperadas, lo que lleva a una mayor confiabilidad de los equipos y a una mejor utilización de los recursos operativos (Ayala, 2017, p.11).

2.2.1.1 Modelos Estadísticos

Los modelos estadísticos describen relaciones entre variables observadas mediante supuestos probabilísticos explícitos. A diferencia de los algoritmos puramente predictivos, estos modelos formulan una función de verosimilitud y parámetros con interpretación, permitiendo inferencia, estimación de incertidumbre y pruebas de hipótesis. En mantenimiento y confiabilidad, se emplean para cuantificar el tiempo hasta la falla, la probabilidad de supervivencia y el efecto de covariables operativas.

Según Ashraf y Khan (2021) estos tipos de modelos están compuestos por los siguientes componentes:

- **Variable respuesta**: continua, binaria, conteo o “tiempo-a-evento”.
- **Covariables**: mediciones del proceso (condición, carga, ambiente).

- **Estructura estocástica**: distribución asumida para la respuesta (Normal, Binomial, Poisson, Weibull, Log-Normal).
- **Vínculo/función del sistema**: cómo las covariables modifican parámetros de la distribución (media, tasa, escala).
- **Parámetros**: estimados por Máxima Verosimilitud (MV) o Bayes.
- **Medidas de ajuste**: AIC/BIC, pruebas de bondad de ajuste, residuos, validación cruzada.
- **Incertidumbre**: errores estándar, intervalos de confianza, bandas de supervivencia.

Cuando el objetivo es modelar tiempo hasta la falla con posibles observaciones censuradas (p. ej., equipos que no fallaron durante el periodo de observación), se utilizan modelos de análisis de supervivencia. Elementos clave:

- **Supervivencia**: $S(t) = P(T > t)$
- **Función de riesgo (hazard)**: $h(t)$, tasa instantánea de falla.
- **Censura**: derecha (más común), izquierda o por intervalo.
- **Verosimilitud con censura**: combina términos de densidad $f(t)$ para fallas observadas y supervivencia $S(t)$para observaciones censuradas.

A. Familia AFT (Accelerated Failure Time)

En los modelos AFT, las covariables aceleran o desaceleran el "reloj" del tiempo a la falla. La forma canónica es:

$$\ln(T) = \beta_0 + \beta^T x + \sigma\varepsilon$$

donde ε sigue una distribución especificada (Weibull, Log-Logística, Log-Normal, etc.). Una variación en $x_j$multiplica el tiempo a la falla por un factor de aceleración $exp(\beta_j)$

- $exp(\beta_j) > 1$: tiempos esperados más largos (menor riesgo).
- $exp(\beta_j) < 1$: tiempos más cortos (mayor riesgo).

B. Weibull-AFT (paramétrico)

El Weibull-AFT asume que T sigue una distribución Weibull con parámetros de escala $\lambda(x)$. En la parametrización AFT:

$$\ln(T) = \beta_0 + \beta^T x + \sigma\varepsilon, \quad \varepsilon \sim Gumbel\ (EV)$$

lo que induce $T \sim Weibull(k, \lambda)$ con:

- $k = {}^{1}/_{\sigma}$ (controla si el riesgo crece, decrece o se mantiene)
- $\lambda = \exp(\beta_0 + \beta^T x)$ (escala dependiente de covariables).

Propiedades prácticas:

- Interpretabilidad directa via factor de aceleración $\exp(\beta_j)$
- **Flexibilidad de riesgo**:
  - $k < 1$ riesgo decreciente (fallas tempranas),
  - $k = 1$ riesgo constante (exponencial),
  - $k > 1$ riesgo creciente (desgaste).
- **Manejo natural de censura** (derecha, intervalo).
- **Estimación de RUL** (vida útil remanente) a partir de la distribución condicional $T|x$

Estimación y validación:

- **Estimación**: Máxima Verosimilitud con censura; intervalos por método asintótico o bootstrap.
- **Bondad de ajuste**: AIC/BIC, gráficos prob-Weibull, residuos de Cox-Snell y devianza.
- **Desempeño predictivo**: C-index (discriminación), Brier score (calibración), curvas ROC-tiempo y calibration plots.

Ventajas en monitoreo de motores eléctricos:

- Integra señales resumidas (RMS, curtosis, espectro, temperatura, corriente) como covariables que modulan el tiempo a la falla.
- Permite cuantificar efectos: p. ej., un incremento de vibración que reduce el tiempo esperado a la falla en un factor < 1.
- Facilita proyecciones individuales de RUL con intervalos de confianza, útiles para planificación de mantenimiento.

2.2.1.2 Machine Learning

El aprendizaje automático (machine learning) es una disciplina perteneciente a la inteligencia artificial que tiene como propósito proporcionar a una máquina la capacidad de aprender y mejorar a partir de datos, mediante el uso de algoritmos, sin necesidad de ser

programada de forma explícita. Este enfoque simula la habilidad humana de aprender a través de ejemplos, sin requerir fórmulas exactas o relaciones definidas entre variables. Al finalizar el proceso de entrenamiento, se genera un modelo capaz de generalizar, es decir, de producir resultados en escenarios nuevos que no fueron contemplados durante el aprendizaje. Esta capacidad es especialmente útil para abordar problemas complejos en los que no se dispone de una fórmula precisa para obtener respuestas a partir de ciertas variables. Por ejemplo, un modelo entrenado con una colección de imágenes y sus características podrá luego identificar y clasificar nuevas imágenes basándose en lo aprendido (Véliz, 2018).

El aprendizaje automático es una disciplina de la computación que proporciona a la Inteligencia Artificial la habilidad de adquirir conocimientos y ejecutar tareas. Para ello, los desarrolladores utilizan algoritmos propios del aprendizaje automático. Entre estos algoritmos se encuentran los denominados árboles de decisión (Véliz, 2018, p.32).

El aprendizaje, al igual que la noción previamente mencionada de inteligencia, se describe como la adquisición de saberes, la comprensión de algún tema o el desarrollo de destrezas para realizar una tarea. El aprendizaje automático representa la base esencial de la inteligencia artificial, y se entiende también como la disciplina que analiza algoritmos computacionales y esquemas estadísticos que permiten que un sistema mejore de manera autónoma mediante la experiencia y el razonamiento (Yon y Luis, 2021).

El aprendizaje de máquinas, también conocido como aprendizaje automatizado o automático, es una subárea de la inteligencia artificial que permite aplicar métodos para que los sistemas informáticos adquieran conocimientos. Esto se realiza mediante el uso de algoritmos y procedimientos que transforman conjuntos de datos en programas, sin requerir que todo el código sea definido manualmente (Albornoz, 2021).

“Existen dos clases de aprendizaje automático: el aprendizaje automático supervisado y el aprendizaje automático no supervisado” (Albornoz, 2021, p.10).

En el Machine Learning Supervisado, es necesaria la participación de una persona para su ejecución, ya que los algoritmos procesan la información a partir de datos previamente organizados, categorizados y etiquetados manualmente. Los datos que se ingresan en el sistema pueden corresponder a tareas de clasificación, cuando se asigna un

elemento a una categoría específica; o a regresión, cuando se busca estimar un valor numérico (Albornoz, 2021).

En el Machine Learning No Supervisado, no se requiere participación humana para su funcionamiento, ya que los algoritmos identifican patrones o vínculos presentes en los datos ingresados, los cuales no están etiquetados. Los métodos utilizados en este tipo de aprendizaje incluyen: agrupamiento (clustering), donde los resultados se organizan en conjuntos similares; y asociación, en la cual se extraen reglas a partir de las relaciones dentro del conjunto de datos. El aprendizaje automático emplea algoritmos para replicar ciertas capacidades de la inteligencia humana. El aprendizaje profundo, por ejemplo, permite representar una red de neuronas artificiales. Estas redes neuronales reproducen de manera aproximada la configuración biológica del sistema nervioso humano (Albornoz, 2021, p.13).

2.2.1.3 Algoritmos de Machine Learning

Los algoritmos de aprendizaje automático desarrollan un modelo matemático a partir de datos utilizados para el entrenamiento, con el fin de realizar predicciones o tomar decisiones sin requerir una programación explícita. Esto permite mejorar tareas que, mediante métodos tradicionales, no lograrían un procesamiento eficiente. Tom M. Mitchell formuló una definición más precisa sobre los algoritmos del aprendizaje automático: Un software aprende a partir de una experiencia E, respecto a una tarea T, y con una métrica de desempeño R, si su efectividad medida por R en relación a la tarea T mejora a medida que se incrementan las experiencias (Yon y Luis, 2021, p.18).

Debido al método de entrenamiento, también guarda relación con la estadística, aunque se diferencian por su enfoque. Mientras que la estadística se orienta a extraer conclusiones a partir de una muestra, el aprendizaje automático se centra en descubrir patrones que puedan ser utilizados para hacer predicciones generalizables. La meta de un sistema de Machine Learning es ser capaz de responder con precisión ante situaciones nuevas, luego de haber adquirido conocimiento derivado de experiencias pasadas (datos de entrenamiento) (Yon y Luis, 2021 p. 19). Por ejemplo, si un sistema que juega ajedrez mejora su nivel y complejidad tras analizar múltiples partidas, se podría afirmar que ha aprendido. De igual modo, si un sistema de reconocimiento de voz mejora su rendimiento al procesar varias muestras de voces distintas, se puede considerar que ha cumplido su propósito. Los métodos estadísticos que permiten que los sistemas informáticos adquieran conocimiento a

partir de la experiencia forman parte de los algoritmos que proporciona el aprendizaje automático o machine learning (Albornoz, 2021, p.25).

Entre las clases de algoritmos que ofrece el aprendizaje automático, ya sea supervisado o no supervisado, se incluyen los siguientes:

- Regresión lineal
- Regresión polinómica
- Árboles de decisión
- Regresión logística
- XGBoost (Extreme Gradient Boosting)
- Bosques Aleatorios
- Redes neuronales
- Máquinas de vectores de soporte (SVM) (Albornoz, 2021, p. 25).

2.2.2 Árboles de decisión

Se trata de modelos que operan mediante la división repetitiva de un conjunto de datos, lo cual genera subconjuntos o patrones conocidos como nodos, los cuales, al ser representados de forma visual, adoptan una estructura similar a la de un árbol. Estos métodos, que utilizan reglas del tipo "si entonces" y cuyo enfoque heurístico se vincula con el conocido refrán "divide y vencerás", fueron desarrollados por Breiman y colaboradores en 1984 (Véliz, 2018, p. 4).

Los árboles de decisión son modelos predictivos que permiten utilizar variables independientes tanto numéricas como cualitativas. Estos modelos son resistentes a valores extremos y no se ven afectados por transformaciones monótonas aplicadas a dichas variables. Cuando la variable objetivo es categórica, se trata de un árbol de clasificación; en cambio, si la variable es continua, el modelo corresponde a un árbol de regresión (Véliz, 2018, p. 5).

Los algoritmos basados en árboles ofrecen una herramienta sencilla, comprensible y eficaz para resolver problemas tanto de regresión como de clasificación. Su principio central consiste en dividir el espacio de características, que puede ser complejo, en subregiones más pequeñas, y en cada una de ellas ajustar una función de predicción sencilla. Por ejemplo, en un caso de regresión, se puede utilizar el promedio de las respuestas de entrenamiento correspondientes a los datos que se encuentran dentro de una determinada región. En un contexto de clasificación, una estrategia común es asignar la categoría más frecuente (voto

mayoritario) entre las variables de salida dentro de esa región. A continuación, se presenta un ejemplo básico de clasificación (Kroese et al., 2020).

Los árboles de decisión, introducidos por Morgan y Sonquist (1963), constituyen una técnica utilizada para generar modelos interpretables de decisión a partir de una muestra de datos disponible. El término modelo hace referencia a que estas técnicas construyen una representación, hipótesis o estructura que refleja las regularidades o patrones presentes en los datos. La palabra interpretables implica que dichos modelos pueden ser expresados simbólicamente, mediante un conjunto de condiciones o reglas descritas en lenguaje verbal (a diferencia de otros métodos más complejos, como las redes neuronales), lo que permite que sean fáciles de entender tanto para las personas como para sistemas automáticos o semiautomáticos que operan con reglas definidas (Camborda, 2014, p. 84).

Un árbol de decisión es un modelo predictivo empleado dentro del campo de la inteligencia artificial. A partir de una base de datos, se generan diagramas con estructuras lógicas, muy parecidos a los sistemas de predicción por reglas, que permiten representar y clasificar una secuencia de condiciones encadenadas, con el propósito de resolver un determinado problema (Camborda, 2014, p. 85).

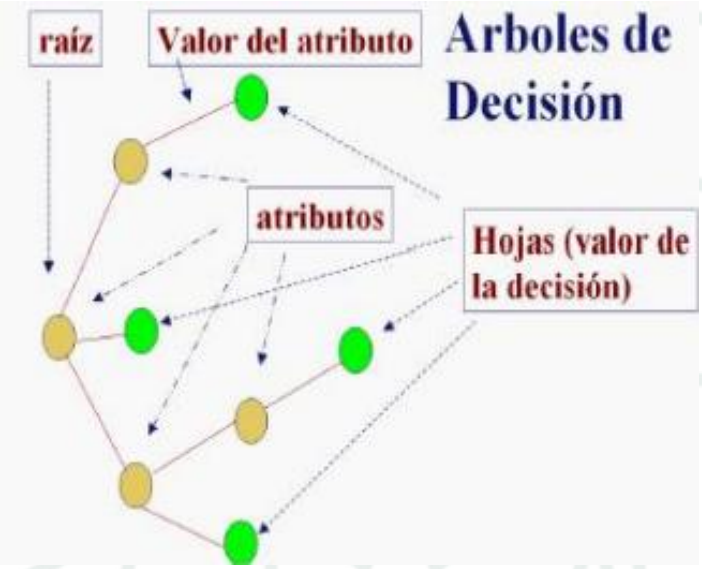
Un árbol de decisión recibe un conjunto de entradas, que pueden representar un objeto o una situación, descritos mediante diversos atributos. A partir de esta información, el modelo genera una salida, que en esencia corresponde a una decisión basada en los datos de entrada. Las entradas y salidas pueden adoptar valores discretos o continuos. Generalmente, se emplean más los valores discretos debido a su facilidad de manejo. Cuando en una aplicación se usan valores discretos, el proceso se conoce como clasificación, mientras que, si se trabajan con valores continuos, se denomina regresión (Camborda, 2014).

Un árbol de decisión realiza una evaluación a medida que se avanza por sus ramas hasta llegar a una conclusión final. Este tipo de árbol suele estar compuesto por nodos internos, nodos probabilísticos, nodos terminales (hojas) y enlaces (arcos). Un nodo interno incluye una prueba o condición relacionada con alguna característica específica. Un nodo de probabilidad señala que debe ocurrir un evento aleatorio conforme a la naturaleza del problema; este nodo se representa de forma circular, mientras que los otros nodos son

cuadrados. Un nodo hoja indica el resultado o salida que proporciona el árbol, y las ramas muestran las distintas rutas posibles según la elección realizada en cada punto del recorrido.

Figura 1

Propiedades de un Árbol de Decisión



Nota. Tomado de (Moreno, 2008).

En el desarrollo de soluciones informáticas, un árbol de decisión representa las acciones a ejecutar según los valores que adopten una o varias variables. Es una estructura en forma de árbol, cuyas ramificaciones se dividen dependiendo de los valores que toman dichas variables, y finalizan en una determinada acción. Este recurso suele emplearse cuando la cantidad de condiciones no es muy extensa; de lo contrario, es preferible utilizar una tabla de decisión (Camborda, 2014).

De manera más específica, dentro del contexto empresarial, los árboles de decisión pueden definirse como diagramas que representan decisiones secuenciales y sus posibles consecuencias. Estas herramientas permiten a las organizaciones analizar sus alternativas, mostrando las distintas opciones disponibles junto con sus resultados esperados. Aquella alternativa que evita una pérdida o genera una ganancia adicional adquiere un valor económico. Por lo tanto, la capacidad de generar una alternativa viable también posee un valor que puede ser negociado o intercambiado (Moreno, 2008, p. 20).

- **XGBoost (Extreme Gradient Boosting):** XGBoost (Extreme Gradient Boosting) es un algoritmo de machine learning de aprendizaje supervisado que implementa la técnica de gradient boosting optimizada para lograr alta precisión y eficiencia computacional. El gradient boosting se basa en la construcción secuencial de múltiples modelos débiles, generalmente árboles de decisión de poca profundidad, donde cada nuevo modelo corrige los errores residuales cometidos por los anteriores (Moreno, 2008, p. 25).

Entre sus principales características destacan:

- **Optimización regularizada:** Incorpora términos de regularización L1 y L2 en su función objetivo para controlar la complejidad de los árboles y reducir el riesgo de sobreajuste (*overfitting*).
- **Entrenamiento paralelo:** Aprovecha múltiples núcleos de CPU para acelerar la generación de árboles.
- **Manejo de datos faltantes:** Utiliza rutas de división predeterminadas (*default directions*) para procesar valores ausentes sin imputación previa.
- **Escalabilidad:** Permite procesar grandes volúmenes de datos mediante estructuras de bloques y compresión de memoria.
- **Versatilidad:** Admite diversas funciones objetivo, incluyendo clasificación binaria, multiclase, regresión y ranking.

Este algoritmo es utilizado por su capacidad de modelar relaciones no lineales, manejar variables heterogéneas y ofrecer resultados precisos en tareas predictivas, incluso con datos de alta dimensionalidad y estructuras complejas.

- **Características de los árboles de decisión:** Cada nodo del árbol representa una característica, y cada ramificación indica un valor posible de dicha característica. Una hoja al final del árbol muestra el resultado previsto de la decisión tomada. La justificación de una decisión específica se obtiene siguiendo el camino que va desde el nodo inicial o raíz hasta la hoja final, donde se concreta el resultado. Este recorrido permite interpretar cómo las distintas condiciones evaluadas a lo largo de la estructura conducen a una determinada conclusión, facilitando así la comprensión del

proceso de toma de decisiones en función de los datos analizados (Moreno, 2008, p. 9).

2.2.3 Arquitectura embebida

La arquitectura embebida hace referencia a la estructura interna de los sistemas embebidos, los cuales son equipos electrónicos programables que integran un procesador, memoria y elementos de entrada y salida para ejecutar tareas definidas. A diferencia de los ordenadores de uso general, estos sistemas están diseñados para realizar funciones concretas, como controlar un electrodoméstico, regular la temperatura en un entorno industrial o procesar datos de sensores. Su estructura suele basarse en microcontroladores o microprocesadores de bajo consumo y desempeño estable. Incluyen una unidad de procesamiento central (CPU), memoria de solo lectura o tipo Flash para guardar instrucciones, memoria de acceso aleatorio (RAM) para datos temporales, y memorias no volátiles como EEPROM. También cuentan con canales de comunicación como UART, SPI o I2C, módulos para contar o medir tiempos, conversores de señales analógicas a digitales, y algunas veces conexiones como USB o salidas PWM. Todos estos elementos están reunidos en un solo chip, permitiendo un funcionamiento eficaz en equipos que requieren respuestas rápidas. Esta estructura es común en dispositivos utilizados en sectores como la manufactura, el control electrónico y los sistemas integrados (Salas, 2015, p. 51).

La arquitectura embebida define la estructura interna de un sistema embebido, el cual está diseñado para realizar funciones específicas dentro de un dispositivo mayor. Estos sistemas combinan procesamiento, memoria y periféricos en una sola unidad compacta (León y Velarde, 2015, p. 30).

Generalmente, se basan en microcontroladores o microprocesadores que incluyen una CPU, memoria de programa (ROM o Flash), memoria de datos (RAM), y periféricos de entrada/salida. Además, pueden integrar módulos de comunicación como UART, SPI o I2C y temporizadores para tareas de control (León y Velarde, 2015, p. 31).

Estos componentes se organizan de forma que permiten una operación continua, eficiente y en tiempo real, adecuada para aplicaciones donde se requiere respuesta inmediata y bajo consumo energético. Su diseño responde a requerimientos concretos y no a funciones generales. La arquitectura embebida es común en sectores como la automatización industrial,

el transporte, la salud, y los sistemas electrónicos de consumo, donde la confiabilidad y la estabilidad del funcionamiento son aspectos prioritarios (León y Velarde, 2015, p. 33).

2.2.4 Estado operativo

El estado funcional de un motor eléctrico hace referencia a las condiciones bajo las cuales opera dentro de un entorno industrial. Este estado comprende variables como temperatura, vibraciones, intensidad eléctrica, tensión y velocidad de rotación. Supervisar estos indicadores permite detectar desviaciones que pueden ser señales tempranas de fallas inminentes, contribuyendo a evitar paradas imprevistas del sistema. La evaluación del estado funcional es esencial para implementar mantenimiento anticipado. Mediante la detección temprana de comportamientos atípicos, se pueden planificar acciones correctivas antes de que ocurran daños severos. Esto favorece la disponibilidad continua del equipo, alarga la vida útil de sus componentes y reduce considerablemente los gastos operativos (León y Velarde, 2015, p. 35).

Con el uso de sistemas embebidos distribuidos, es posible realizar un seguimiento continuo del estado funcional mediante sensores inteligentes. Estos dispositivos recopilan datos en tiempo real y los transmiten utilizando protocolos ligeros como MQTT. Esta forma de envío permite consolidar información sin sobrecargar la red, mejorando la capacidad de respuesta en la gestión operativa (León y Velarde, 2015, p. 36). El registro constante de las condiciones del motor permite identificar tendencias de comportamiento. Esto es útil para crear modelos predictivos que reconozcan patrones de desgaste o deterioro. Así, se puede anticipar la necesidad de intervención, asegurando una mejor planificación de los recursos técnicos (Centellas y Carlos, 2020, p. 6).

Por tanto, analizar el estado funcional implica ir más allá del simple funcionamiento del motor. Se trata de estudiar en profundidad su rendimiento y comportamiento para garantizar operaciones confiables y eficientes, especialmente en entornos automatizados donde una falla puede tener impactos críticos en la producción (Centellas y Carlos, 2020, p. 15).

2.2.5 Degradación de motores eléctricos

La degradación de un motor eléctrico se refiere al deterioro progresivo de sus componentes internos, debido principalmente a condiciones de operación no ideales, envejecimiento de materiales y mantenimiento inadecuado. Esta degradación puede manifestarse en diversas formas:

- Desgaste del aislamiento de los devanados, provocado por sobrecalentamiento, humedad o contaminación.
- Desequilibrios eléctricos o mecánicos, que generan vibraciones, calentamiento irregular y fallas en los rodamientos.
- Fatiga de materiales, especialmente en partes móviles como el rotor o el sistema de acoplamiento.
- Contaminación interna, causada por polvo, humedad, aceite o químicos que ingresan al motor, afectando su ventilación y aislamiento.

  (Centellas y Carlos, 2020, p.19).

2.2.5.1 Principales causas de Degradación

- **Fatiga térmica del aislamiento:** La exposición prolongada a altas temperaturas reduce la resistencia dieléctrica de los materiales aislantes, lo que genera cortocircuitos o fallas entre espiras.
- **Vibraciones mecánicas:** Desalineaciones, desequilibrios o desgaste en cojinetes generan vibraciones que aceleran el deterioro de partes móviles y estructuras internas.
- **Contaminación ambiental:** La presencia de polvo, humedad, productos químicos o aceite reduce la eficacia del sistema de enfriamiento y puede dañar el aislamiento o provocar corrosión.
- **Sobrecargas eléctricas:** El funcionamiento continuo por encima de la potencia nominal produce calentamiento excesivo y reduce la vida útil del motor.
- **Fallos en lubricación:** La degradación o ausencia de lubricante en los rodamientos incrementa el roce, generando calor, ruido y desgaste prematuro (Holguín y Álvarez, 2019, p. 69).

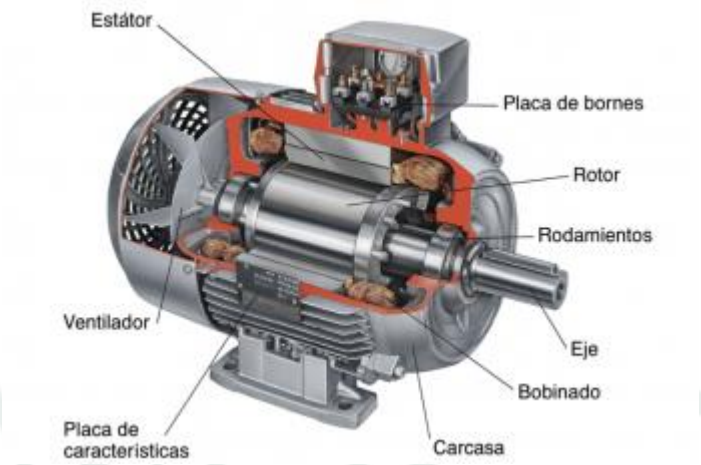
2.2.6 Motores eléctricos

Los motores eléctricos poseen una vida útil estimada entre 15 y 20 años, operando entre 4000 y 6000 horas anuales en entornos industriales. En el caso de los motores utilizados en la UPS, su uso promedio es de aproximadamente 6 horas por semana durante las prácticas relacionadas con maquinaria e instalaciones industriales. Como resultado, experimentan un deterioro gradual en sus componentes mecánicos, lo que genera fallos que acortan su tiempo de funcionamiento (Tene y Ortiz, 2016, p. 30).

"Una máquina eléctrica es un equipo capaz de transformar la energía mecánica en energía eléctrica, o realizar la conversión inversa. En la Figura 2 se muestran los componentes que integran un motor eléctrico" (Chapman, 2012, p. 1, p. 84).

Figura 2

Partes de un motor eléctrico



Nota. Tomado de (Tene y Ortiz, 2016).

2.2.6.1 Estator

"Es el componente esencial de un motor en el que se genera el campo magnético necesario para provocar el movimiento del propio estator" (Pinto, 2019, p.15).

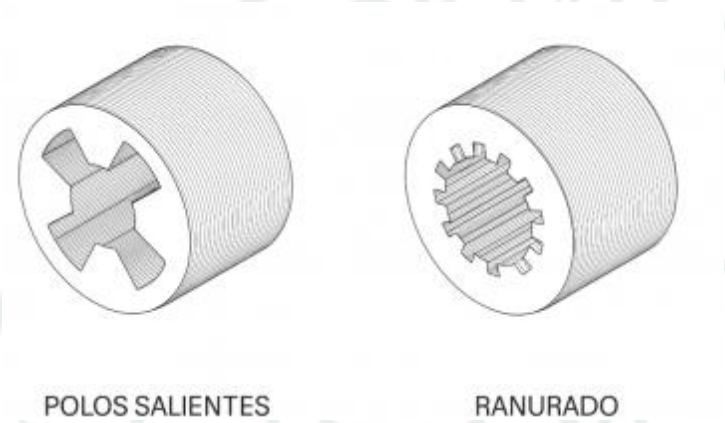
"El estator está constituido por una serie de láminas de acero al silicio, que junto con los bobinados crean los polos magnéticos del motor" (Pinto, 2019, p.15).

La disposición de estos polos magnéticos sigue un orden par, siendo el número mínimo de polos dos (norte y sur). Este elemento puede encontrarse en dos configuraciones distintas:

- Estator con polos salientes
- Estator ranurado. (Pinto, 2019, p.15)

Figura 3

Tipos de Estator



Nota. Elaboración propia.

2.2.6.2 Rotor

Es el componente del motor donde se realiza la transformación de energía eléctrica en energía mecánica. Al igual que el estator, está formado por chapas de acero al silicio. Existen tres clases principales de rotores que se pueden identificar:

- Eje con rotor con ranuras
- Eje con rotor de polos sobresalientes
- Eje con rotor tipo jaula de ardilla (Pinto, 2019, p.16).

Figura 4

Tipos de Rotor

Nota. Elaboración propia.

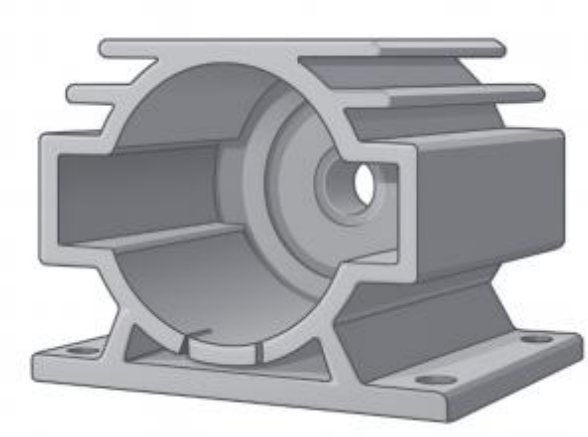
2.2.6.3 Carcasa

Está compuesta por la estructura resistente que resguarda todo el motor. El tipo de material y el diseño de fabricación dependen del modelo de motor, así como de su uso o forma de operación (Pinto, 2019). Según estas características, las envolventes pueden clasificarse en distintas variantes:

- Carcasa completamente sellada
- Carcasa ventilada
- Carcasa resistente al goteo
- Carcasa con protección contra explosiones
- Carcasa diseñada para inmersión

Figura 5

Carcasa del motor



Nota. Elaboración propia.

“Cuando este equipo se emplea para transformar energía mecánica en energía eléctrica, se le conoce como generador; mientras que, si realiza la conversión de energía eléctrica en mecánica, se denomina motor” (Chapman, 2012, p. 10).

Un motor eléctrico no requiere combustibles ni sistemas de ventilación para funcionar, a diferencia de los motores de combustión interna, que sí los necesitan. Por esta razón, los motores eléctricos se prefieren en lugares donde se desea evitar la generación de residuos contaminantes (Chapman, 2012, p. 10).

2.2.6.4 Clasificación

“Los motores eléctricos se dividen según el tipo de corriente que utilizan como fuente de energía, clasificándose en: Motores de corriente directa y Motores de corriente alterna” (Tene y Ortiz, 2016).

a. Motores de corriente directa

Los primeros sistemas de generación eléctrica en Estados Unidos funcionaban con corriente directa, pero hacia finales de la década de 1890 se evidenció que los sistemas basados en corriente alternan comenzaban a imponerse. No obstante, los motores de corriente continua continuaron representando una proporción relevante de la maquinaria adquirida anualmente hasta los años 60 (Chapman, 2012, p.47).

Los motores de corriente continua actúan como generadores que convierten la energía mecánica en electricidad de tipo continua, y también como motores que transforman la energía eléctrica continua en energía mecánica. Estas máquinas guardan gran semejanza con las de corriente alterna, ya que en su interior manejan tanto voltajes como corriente alterna. Sin embargo, los motores de corriente continua proporcionan una salida de corriente continua gracias a un sistema que transforma los voltajes internos de corriente alterna en voltajes de corriente directa en sus terminales. Por esta razón, también se conocen como máquinas de conmutación (Chapman, 2012, p. 48).

Los conceptos básicos que rigen el funcionamiento de las máquinas de corriente directa son relativamente sencillos y fáciles de comprender. Sin embargo, en muchas

ocasiones esta simplicidad queda opacada por la complejidad inherente al diseño, fabricación y ensamblaje de los equipos reales. A pesar de que el principio operativo es directo, los detalles técnicos implicados en su construcción como la precisión en el ensamblaje, la selección de materiales adecuados y la configuración interna suelen generar confusión o dificultar su entendimiento general. Por ello, es común que la dificultad estructural de estos sistemas oculte la claridad de su funcionamiento esencial (Chapman, 2012, p. 49).

b. Motores de corriente alterna

Estos tipos de motores operan utilizando esta clase de suministro eléctrico, aunque su mecanismo de acción es similar al de un motor de corriente continua, ya que ambos generan un movimiento giratorio impulsado por la influencia de un campo magnético. La conversión de energía eléctrica en energía mecánica ocurre gracias a la interacción entre las corrientes eléctricas y el campo magnético dentro del motor, lo que origina una fuerza que provoca la rotación del eje. Esta característica permite su aplicación en múltiples sistemas industriales y domésticos (Tene y Ortiz, 2016, p. 69).

Entre los tipos más representativos de motores que trabajan bajo este principio se encuentran:

- Motor monofásico, que utiliza una sola fase de corriente alterna y es común en aparatos domésticos y maquinaria ligera.
- Motor Dahlander, diseñado para cambiar la velocidad de giro mediante la modificación de la conexión de los devanados del estator, sin necesidad de emplear variadores externos.
- Motor trifásico, que funciona con tres fases de corriente alterna, proporcionando mayor potencia y eficiencia, siendo ideal para usos industriales donde se requiere un rendimiento elevado y constante (Tene y Ortiz, 2016, p. 70).

- **Motor Monofásico**

Este tipo de motores y generadores son, con gran diferencia, los más empleados en grandes instalaciones industriales y comerciales. No obstante, la mayoría de las viviendas y pequeñas compañías no cuentan con sistemas de alimentación trifásica (Chapman, 2012).

El inconveniente principal en el diseño de los motores de inducción monofásicos radica en que, a diferencia de las fuentes trifásicas, una fuente monofásica no genera un campo magnético giratorio. En su lugar, la fuente monofásica produce un campo magnético que permanece fijo y varía con el tiempo. Debido a la ausencia de un campo magnético rotativo neto, los motores de inducción estándar no pueden operar correctamente, por lo que es necesario implementar configuraciones especiales para su funcionamiento (Chapman, 2012).

- **Motor Dahlander**

El motor Dahlander cuenta en su bobinado con varias derivaciones intermedias que se utilizan exclusivamente para modificar la cantidad de polos en funcionamiento. Esto permite ajustar la velocidad del motor. Es importante considerar que, al contar con dos configuraciones de conexión, se obtienen dos velocidades distintas: una baja y otra alta. La carcasa del motor incluye 9 terminales, correspondientes a esas derivaciones intermedias. Este tipo de motor era comúnmente empleado en elevadores, equipos de izaje y maquinaria industrial. Actualmente, resulta más eficiente y conveniente el uso de variadores de frecuencia, ya que ofrecen el mismo efecto de regulación de velocidad (Bajuli, 2009).

- **Motor trifásico**

Los motores eléctricos trifásicos se fabrican en una amplia variedad de potencias, alcanzando incluso miles de caballos de fuerza (HP), y están diseñados para operar con diferentes niveles de voltaje y frecuencias (50 y 60 Hz) que están estandarizadas. Se trata de una máquina eléctrica de tipo rotativo, capaz de convertir energía eléctrica trifásica en energía mecánica (Bajuli, 2009).

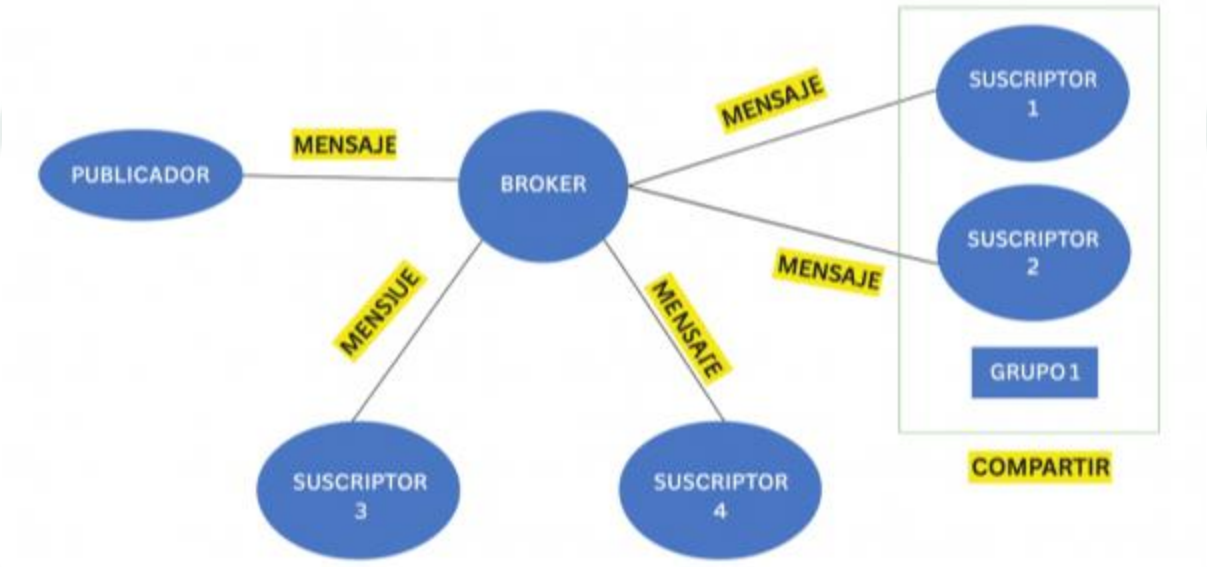
2.2.7 Protocolo MQTT

En entornos con recursos altamente restringidos, como ocurre en el Internet de las Cosas (IoT), es fundamental contar con todas las herramientas posibles para optimizar su aprovechamiento. Por esta razón, existen protocolos denominados livianos, diseñados específicamente para mejorar la eficiencia en el uso de dichos recursos. Estos protocolos operan sobre TCP/UDP e incluso sobre HTTP. Ejemplos representativos de este tipo son MQTT y CoAP, los cuales, gracias a su modelo de comunicación (como se muestra en la

figura 5), logran una mayor eficiencia en comparación con HTTP dentro de sistemas IoT. Desde el año 2018, el esquema de comunicación basado en el modelo publicador/suscriptor, ilustrado en la figura 6, ha pasado a ser el más utilizado, desplazando a protocolos como HTTP y CoAP. El principal aporte de estos protocolos radica en su capacidad para optimizar el consumo de recursos limitados como el procesamiento, el ancho de banda y la energía en dispositivos IoT, lo cual ha sido ampliamente documentado en diversos estudios (Pinzón, 2020).

Figura 6

Comparativa de Reenvío MQTT: Estándar vs Suscripción Compartida

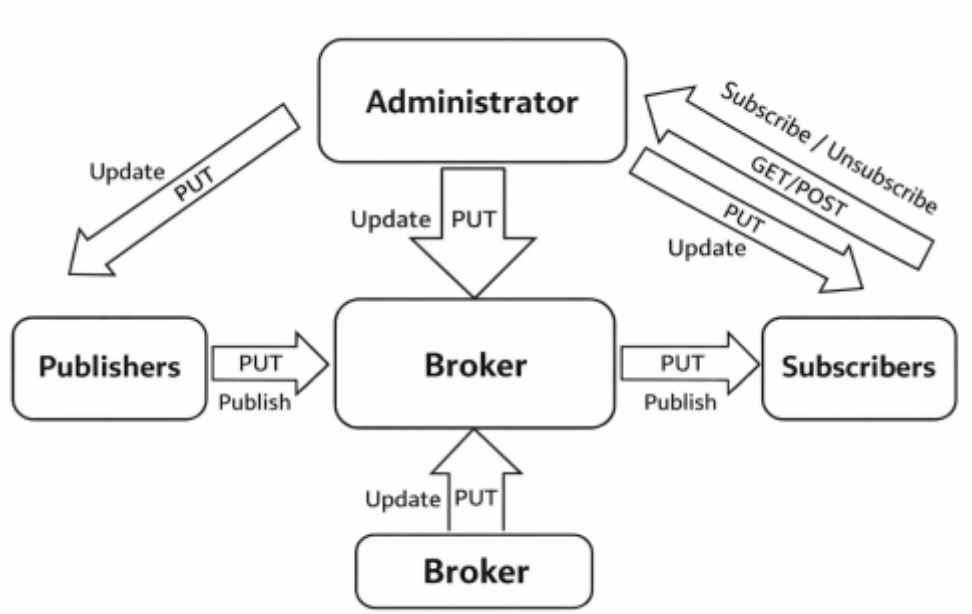


Nota. Tomado de (Milica y Oros, 2020).

Este protocolo representa un sistema de mensajería para el envío de datos telemétricos en cola. Se basa en un esquema de comunicación publicador/suscriptor, pensado para intercambios livianos entre máquinas (M2M). Fue creado inicialmente por IBM y, en la actualidad, se ha convertido en un estándar. La arquitectura de MQTT funciona bajo un modelo Cliente/Servidor, donde cada sensor actúa como cliente y se conecta a un servidor conocido como corredor o broker, utilizando el protocolo de comunicación TCP. Cada mensaje transmitido se publica en un "topic", que es una cadena de texto, y los clientes pueden suscribirse a múltiples topics para recibir la información correspondiente (Nikolov et al., 2020, p. 84).

Figura 7

Modelo Publish/Subscribe con MQTT

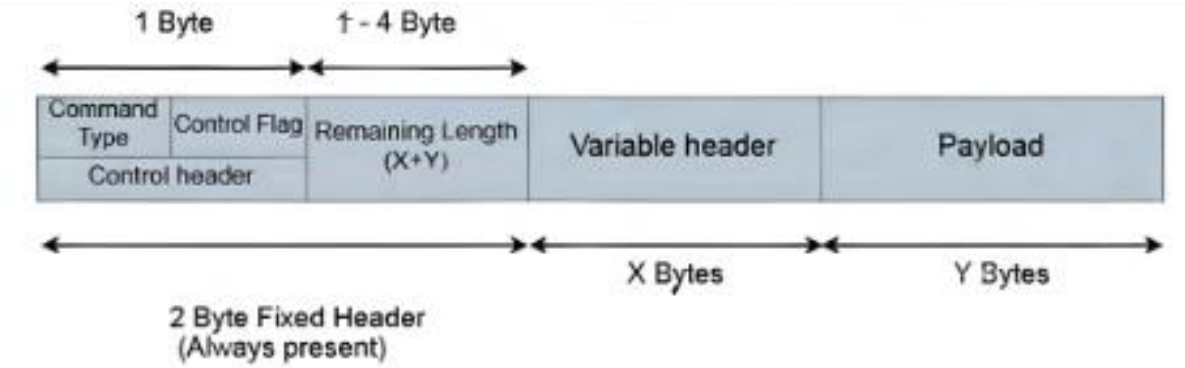


Nota. Tomado de (Pinzón, 2020).

Dentro de los protocolos livianos, MQTT es uno de los más utilizados por su facilidad de implementación y buen rendimiento, gracias a que emplea el esquema publicador/suscriptor. Además, cuenta con una cabecera de solo 2 bytes, lo que representa un tamaño considerablemente menor en comparación con los 137 bytes que requiere HTTP (Pinzón, 2020).

Figura 8

Estructura del paquete MQTT



Nota. Tomado de (Jimenez, 2020).

Como consecuencia, se logra una mayor eficiencia en el uso del ancho de banda y el consumo energético al emplear MQTT, gracias tanto a su modelo de comunicación como al reducido tamaño de su cabecera. Por otro lado, según lo analizado, CoAP presenta un desempeño superior en términos de tiempo de respuesta (RTT) y uso de ancho de banda. No obstante, se recomienda utilizar una combinación de ambos protocolos MQTT y CoAP o, en su defecto, optar únicamente por MQTT para obtener mejores resultados. Esto se debe a que el esquema publicador/suscriptor se adapta mejor a los entornos IoT. En concreto, se sugiere utilizar MQTT para la comunicación entre sensores, mientras que CoAP es más adecuado para enlazar dispositivos más complejos, como los servidores, ya que estos pueden manejar sin dificultad el protocolo HTTP (Jimenez, 2020, p. 90).

CAPÍTULO III

PLANTEAMIENTO DE LA INVESTIGACIÓN

3.1 Metodología de Investigación

La investigación se desarrollará bajo un enfoque cuantitativo, ya que se emplearán datos medibles obtenidos de simulaciones y análisis de métricas para evaluar el desempeño del sistema propuesto. Se utilizará un método experimental-simulativo, el cual consiste en el diseño, modelado y validación virtual de un sistema predictivo con arquitectura embebida distribuida, que incorpore sincronización de datos vía MQTT y modelos de machine learning (XGBoost y Weibull-AFT).

3.2 Tipo de Investigación

La investigación es de tipo experimental, dado que se manipularán variables independientes en entornos simulados para observar y medir sus efectos sobre la variable dependiente. El estudio se desarrollará a nivel de diseño y simulación, utilizando escenarios controlados que permitan evaluar el desempeño del sistema predictivo con arquitectura embebida distribuida y sincronización de datos vía MQTT. El alcance es explicativo, pues busca establecer la relación causa-efecto entre la implementación de la arquitectura propuesta, el uso de modelos predictivos y la mejora en la detección de degradación en motores eléctricos.

3.3 Diseño de Investigación

Se adoptará un diseño experimental-no probabilístico, en el que se crearán entornos controlados de simulación para evaluar el comportamiento del sistema predictivo propuesto. Las pruebas contemplarán la variación de parámetros simulando condiciones de operación y degradación en motores eléctricos. No se realizará implementación física; el estudio se limitará a entornos de simulación que permitirán aislar y medir el impacto de cada variable independiente sobre la variable dependiente.

3.4 Técnicas e Instrumentos

3.4.1 Técnicas

Para el desarrollo de la investigación se emplearán las siguientes técnicas:

a) **Observación indirecta**: Recolección de datos provenientes de simulaciones en entornos virtuales, registrando variables operativas como vibración, temperatura y corriente.
b) **Modelado y simulación computacional**: Utilización de software especializado (MATLAB/Simulink) para diseñar, validar y optimizar la arquitectura embebida distribuida y la comunicación mediante MQTT.
c) **Análisis de datos**: Aplicación de métricas estadísticas y de desempeño (Accuracy, F1, ROC-AUC, RMSE, C-index, Brier score) para evaluar la efectividad de los modelos predictivos XGBoost y Weibull-AFT.

3.4.2 Instrumentos

Los instrumentos que se emplearán son:

a) **Lenguaje Python**: Para modelar el comportamiento de motores eléctricos, la arquitectura embebida distribuida y los protocolos de comunicación.
b) **Colaboratory:** Para la recepción, almacenamiento y visualización de datos generados por los nodos de monitoreo en tiempo real.
c) **Bibliotecas y frameworks de machine learning**: Para procesamiento de datos y ejecución de algoritmos de clasificación y análisis de supervivencia.
d) **Hojas de cálculo:** Para la tabulación y análisis comparativo de resultados obtenidos en las simulaciones.

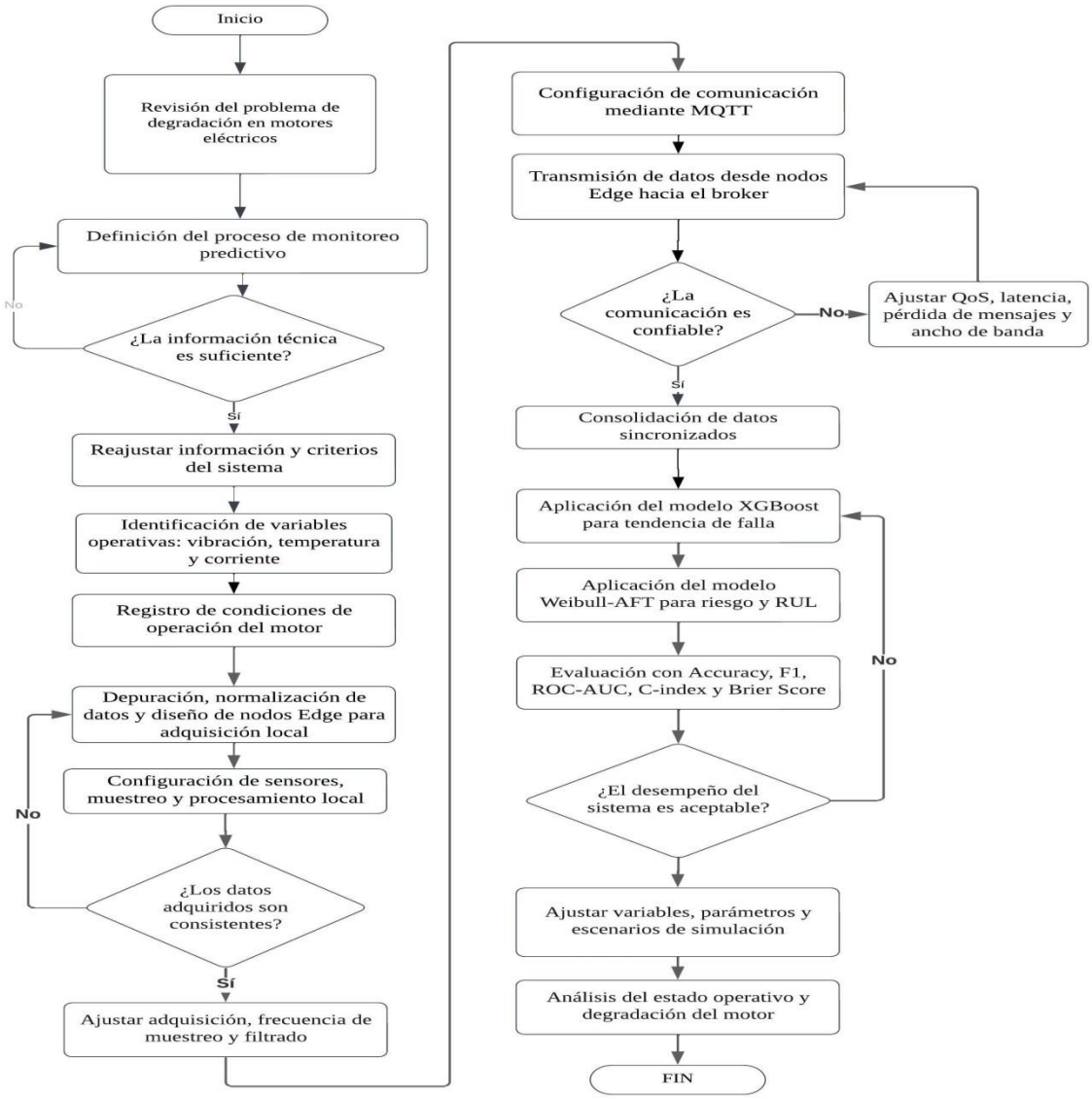
3.5 Esquema metodológico

El diagrama muestra la secuencia metodológica seguida para desarrollar el sistema predictivo aplicado al monitoreo de motores eléctricos. El proceso inicia con la revisión del problema de degradación, continúa con la identificación de variables operativas como vibración, temperatura y corriente, y luego considera la adquisición, depuración y procesamiento local de datos mediante nodos Edge. Posteriormente, se configura la comunicación MQTT para la transmisión y sincronización de la información hacia el broker.

Con los datos consolidados, se aplican los modelos XGBoost y Weibull-AFT para analizar la tendencia de falla, el riesgo y la vida útil remanente. Finalmente, el desempeño del sistema se evalúa mediante métricas como Accuracy, F1, ROC-AUC, C-index y Brier Score, permitiendo validar la coherencia de los resultados obtenidos.

Figura 9

Diagrama de flujo del proceso metodológico



Nota. Elaboración propia

3.6 Campo de Verificación

3.6.1 Ubicación Temporal

La investigación se llevará a cabo durante el año 2025.

3.6.2 Ubicación espacial

El estudio se desarrollará en entornos de simulación virtual y de laboratorio académico. Las simulaciones y pruebas se ejecutarán en equipos de cómputo con software para modelado, comunicación IoT y análisis predictivo.

3.6.3 *Unidades de Estudio*

Las unidades de estudio estarán constituidas por modelos virtuales de motores eléctricos representativos del entorno industrial, configurados para evaluar variables como vibración, temperatura y corriente. Se incluirán nodos de procesamiento “edge” simulados, un broker MQTT para sincronización de datos, y los modelos predictivos XGBoost y Weibull-AFT integrados en la arquitectura embebida distribuida, a fin de analizar su desempeño y capacidad de diagnóstico de degradación.

CAPÍTULO IV

SISTEMA DE MONITOREO Y SINCRONIZACIÓN DE DATOS EN MOTORES ELÉCTRICOS

4.1 Variables operativas que inciden en la degradación de motores eléctricos

El análisis de las variables operativas permite comprender las condiciones reales bajo las cuales un motor eléctrico experimenta desgaste o pérdida de desempeño. Factores como la vibración, la temperatura del devanado, la corriente y la carga influyen directamente en el proceso de degradación de sus componentes mecánicos y eléctricos. El seguimiento continuo de estos parámetros posibilita detectar desviaciones respecto a los valores normales de operación y anticipar posibles fallas antes de que ocurran. En este apartado se presentan los principales indicadores obtenidos a partir de los registros experimentales del sistema de monitoreo, los cuales reflejan el comportamiento promedio del motor durante el periodo de observación. Estos datos permiten establecer una base comparativa para evaluar la severidad de las condiciones de operación y su relación con los límites técnicos establecidos por normas internacionales. Considerando los datos del Anexo N°2, se presenta el siguiente resumen:

La Tabla 1 muestra el resumen de los registros obtenidos durante el periodo de monitoreo del motor eléctrico, comprendido entre el 2 y el 8 de octubre de 2025. En total se recopilaron 150 registros, los cuales reflejan el comportamiento promedio de las principales variables operativas que influyen en su desempeño y degradación.

Tabla 1

Resumen de registro por indicador

Indicador	Valor
Registros	150
Inicio	2025-10-02 16:34:36
Fin	2025-10-08 21:34:36
Vib. media (mm/s)	2.55
Temp. devanado media (°C)	69.06
Corriente media (%FLA)	79.36
Carga media (%)	76.5
ISO A	111
ISO B	29
ISO C	10

Nota. Elaboración Propia

Se observa que la vibración media registrada fue de 2.55 mm/s, valor que se encuentra dentro de la zona A según la norma ISO 20816, indicando condiciones de funcionamiento normales. La temperatura media del devanado alcanzó los 69.06 °C, situándose por debajo del límite de advertencia para motores con aislamiento clase F (105 °C), lo que sugiere una adecuada disipación térmica. En cuanto a la corriente media, se registró un 79.36 % del valor de plena carga (FLA), lo que evidencia una operación estable y sin sobrecargas prolongadas.

La Tabla 2 presenta los límites técnicos de operación establecidos para motores eléctricos, los cuales sirven como referencia para evaluar el estado de funcionamiento y determinar posibles niveles de riesgo asociados a la degradación de los componentes. Estos rangos se basan en estándares internacionales, como la norma ISO 20816, y en especificaciones de diseño propias de motores industriales con potencias comprendidas entre 15 y 300 kW y montaje rígido.

Tabla 2

Límites técnicos de motores eléctricos

variable	umbral	clasificación	nota
Vibración global (mm/s RMS)	≤ 2.8	Zona A (ISO 20816)	Motores 15–300 kW, montaje rígido
Vibración global (mm/s RMS)	(2.8, 4.5]	Zona B (ISO 20816)	Idem
Vibración global (mm/s RMS)	(4.5, 7.1]	Zona C (ISO 20816)	Idem
Vibración global (mm/s RMS)	> 7.1	Zona D (ISO 20816)	Idem
Temp. devanado (°C)	≤ 105	Normal	Clase F, referencia continua
Temp. devanado (°C)	(105, 115]	Observación	Seguimiento cercano
Temp. devanado (°C)	(115, 130]	Alarma	Activar plan de mantenimiento
Temp. devanado (°C)	> 130	Crítica/Paro	Parada programada o inmediata
Corriente (%FLA)	≤ 105%	Normal	Basado en FLA de placa
Corriente (%FLA)	(105, 115]%	Observación	Monitorear factor de potencia/calor
Corriente (%FLA)	(115, 130]%	Alarma	Verificar sobrecarga/armónicos
Corriente (%FLA)	> 130%	Crítica/Paro	Riesgo térmico y de aislamiento

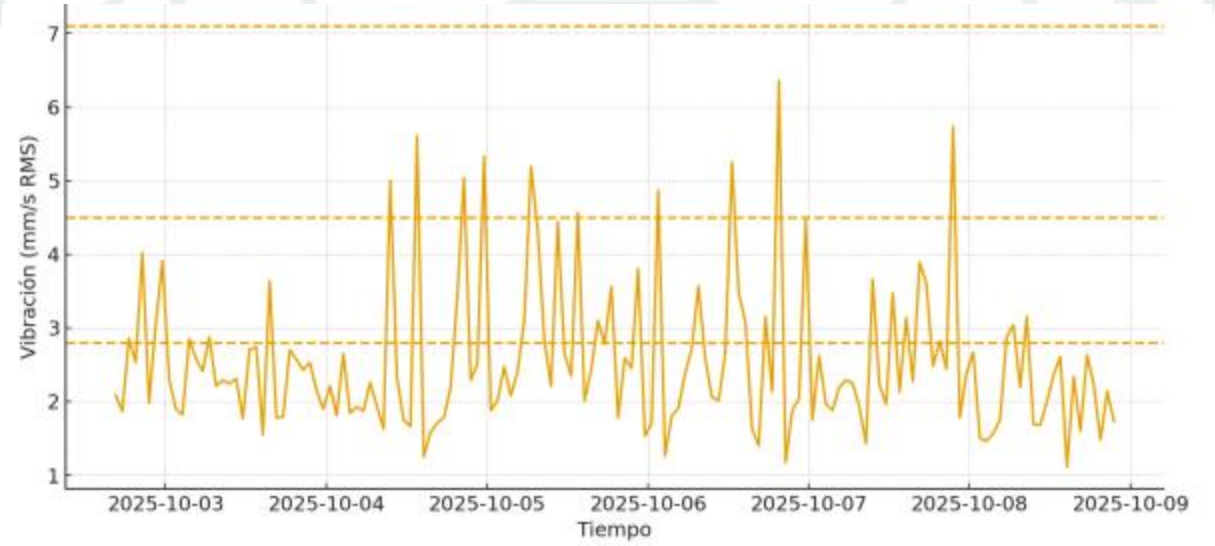
Nota. Elaboración propia a partir de los datos obtenidos en la página de Kaggle

En el caso de la vibración global, los valores menores o iguales a 2.8 mm/s RMS corresponden a la Zona A, que indica condiciones óptimas de funcionamiento. Los valores que se ubican entre 2.8 y 4.5 mm/s pertenecen a la Zona B, donde se recomienda observación periódica, mientras que los rangos entre 4.5 y 7.1 mm/s (Zona C) reflejan una condición de alarma, asociada a un nivel de vibración que puede afectar la integridad de los rodamientos. Valores superiores a 7.1 mm/s (Zona D) son considerados críticos, requiriendo intervención inmediata para evitar fallas severas. En cuanto a la temperatura del devanado, los motores con aislamiento clase F presentan un rango seguro de hasta 105 °C. Superar este valor hasta 115 °C exige un seguimiento cercano, mientras que temperaturas entre 115 °C y 130 °C requieren activar planes de mantenimiento. Niveles mayores a 130 °C representan un estado crítico que justifica la parada inmediata del motor. Por otro lado, la corriente expresada como porcentaje del valor de plena carga (FLA) se considera normal hasta el 105 %, siendo recomendable la observación entre 105 % y 115 %. Una corriente entre 115 % y 130 % indica sobrecarga o presencia de armónicos, y valores superiores al 130 % se clasifican como críticos, ya que incrementan el riesgo térmico y el deterioro del aislamiento.

La Figura 10 muestra la evolución temporal de la vibración global del motor eléctrico durante el periodo comprendido entre el 2 y el 8 de octubre de 2025.

Figura 10

Vibración global en tiempo



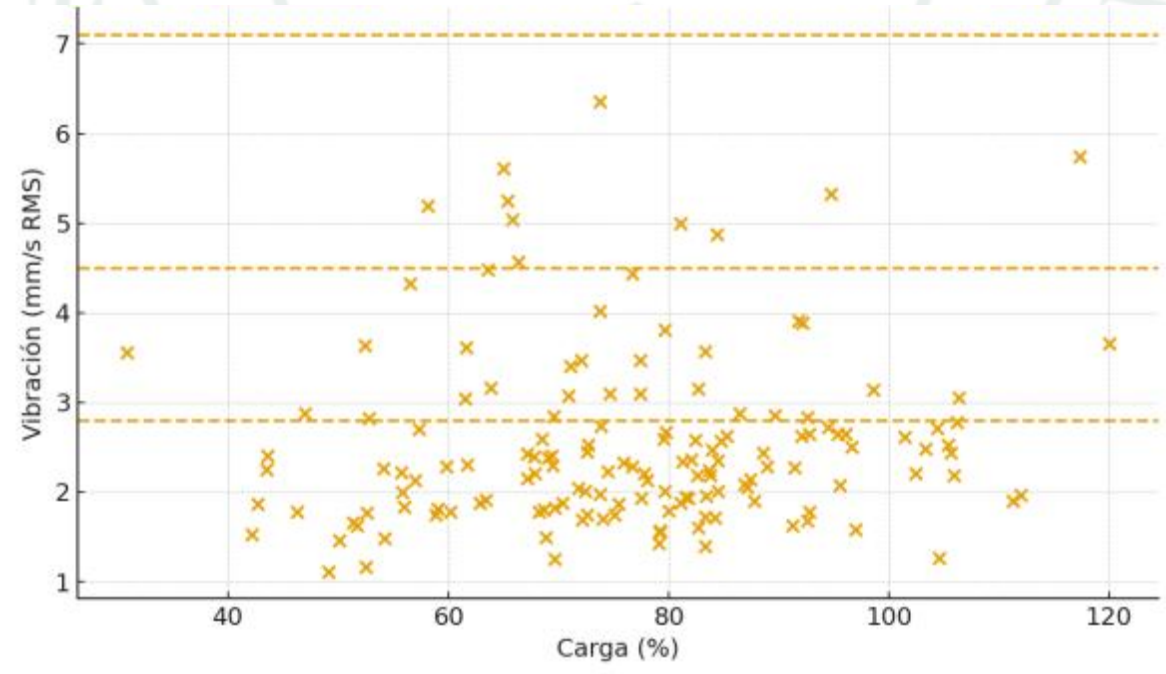
Nota. Elaboración propia

Se observa la variación de la velocidad de vibración en milímetros por segundo (mm/s RMS), contrastada con las líneas de referencia establecidas según la norma ISO 20816, que definen los límites de las zonas A, B, C y D. Durante el periodo analizado, la mayoría de los registros se mantiene dentro de la zona A (≤ 2.8 mm/s), lo que indica un funcionamiento estable y dentro de los parámetros normales. No obstante, se identifican picos aislados que alcanzan valores comprendidos entre 4 y 6 mm/s, correspondientes a la zona B y C, los cuales representan incrementos transitorios en la vibración posiblemente asociados a variaciones de carga o ligeros desbalances en el sistema.

La Figura 11 presenta la relación entre la carga (%) y la vibración global (mm/s RMS) del motor eléctrico durante el periodo de observación. Cada punto representa un registro obtenido, mostrando cómo varía el nivel de vibración conforme cambia la carga aplicada al motor. Se aprecia que la mayoría de los valores de vibración se concentran entre 1.5 y 3 mm/s, incluso cuando la carga oscila entre 60 % y 90 %, lo que sugiere un comportamiento estable dentro de la zona A definida por la norma ISO 20816. Sin embargo, existen algunos puntos aislados que superan los 4.5 mm/s, correspondientes a las zonas B y C, lo cual indica leves incrementos de vibración bajo condiciones de carga más elevadas o transitorias.

Figura 11

Carga (%) vs Vibración (mm/s RMS)

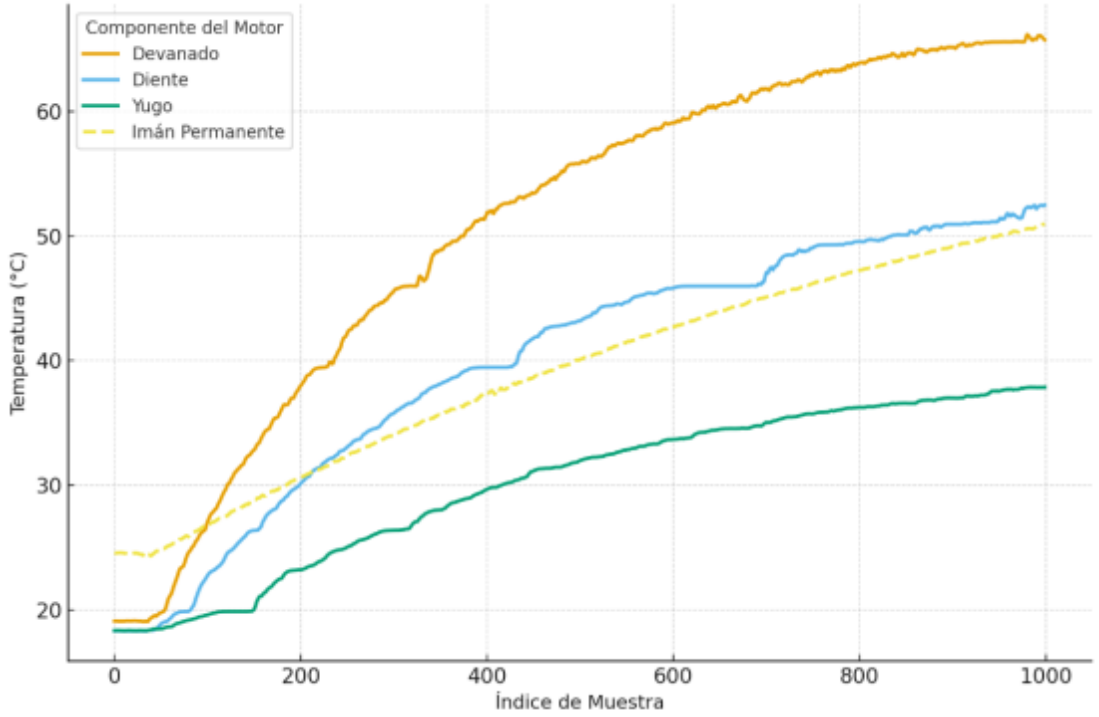


Nota. Elaboración propia

La Figura 12 muestra la evolución térmica interna del motor eléctrico, representando el incremento de temperatura en sus principales componentes devanado, diente, yugo e imán permanente a lo largo del tiempo. Esta información permite analizar el comportamiento térmico del sistema durante el funcionamiento continuo y evaluar su estabilidad frente a las condiciones operativas. Se observa que el devanado alcanza las temperaturas más altas, superando los 60 °C, debido a la acumulación de pérdidas por efecto Joule, lo que es coherente con su función de conducción de corriente. El diente y el imán permanente presentan incrementos moderados, estabilizándose entre 45 °C y 50 °C, mientras que el yugo mantiene los valores más bajos, cercanos a 38 °C, evidenciando una adecuada disipación térmica. El patrón ascendente y gradual de todas las curvas refleja una distribución térmica estable y controlada, sin indicios de sobrecalentamiento. Este comportamiento confirma que el motor opera dentro de los límites establecidos por la clase térmica F (≤ 105 °C), garantizando la integridad del aislamiento y prolongando la vida útil de los materiales.

Figura 12

Evolución térmica interna del motor



Nota. Elaboración propia

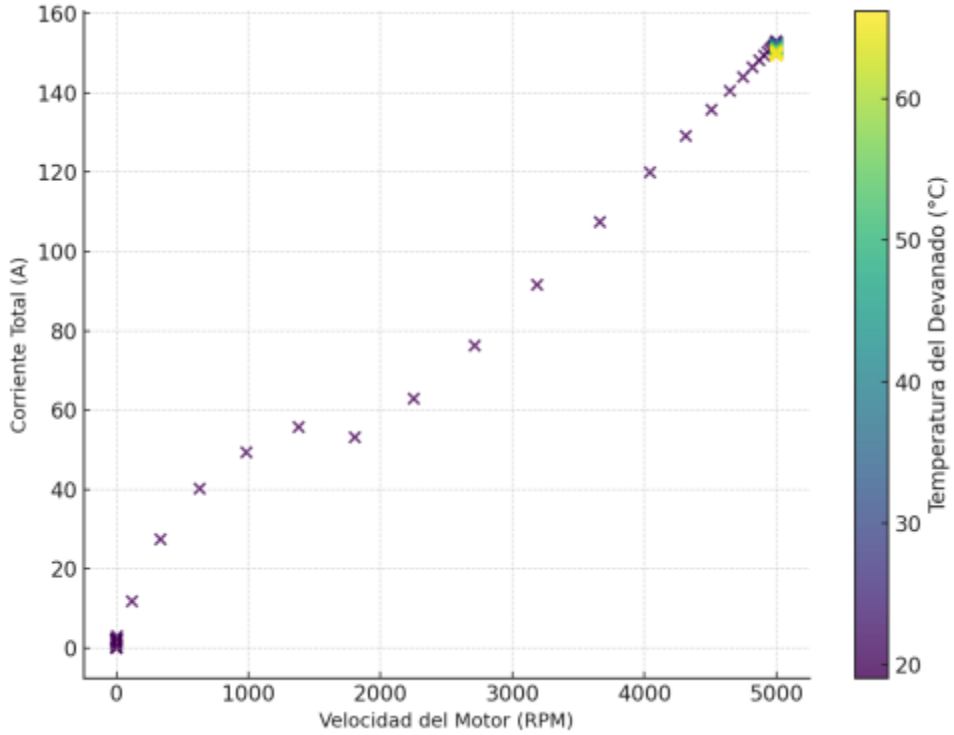
La Figura 13 representa la relación entre la corriente total (A) y la velocidad del motor (RPM), incorporando además una escala de color que indica la temperatura del

devanado (°C). Este gráfico permite analizar cómo la demanda de corriente varía con el incremento de la velocidad y cómo dicha variación influye en el comportamiento térmico del motor. Se observa una relación directamente proporcional entre la corriente y la velocidad: a medida que el motor incrementa su régimen de giro, la corriente aumenta de manera casi lineal hasta alcanzar valores cercanos a 150 A a 5000 RPM. Este comportamiento es característico en motores de inducción, donde el esfuerzo eléctrico crece con la carga y la exigencia mecánica.

El gradiente de color evidencia que, conforme aumenta la corriente, también se eleva la temperatura del devanado, alcanzando máximos en torno a 65 °C. Sin embargo, estos valores se mantienen dentro del rango seguro de operación, lo que indica que existe una adecuada gestión térmica y una correcta capacidad de disipación del calor generado por pérdidas resistivas.

Figura 13

Corriente vs velocidad del motor



Nota. Elaboración propia

4.2 Diseño de arquitectura embebida distribuida para adquisición de datos

La presente sección introduce el esquema general de la arquitectura embebida distribuida desarrollada para la adquisición local de datos en el sistema de monitoreo predictivo. Su finalidad es garantizar que las variables operativas del motor como vibración, temperatura y corriente sean registradas de manera precisa, continua y sincronizada entre los distintos nodos del sistema. El diseño se basa en un enfoque modular y jerárquico, donde cada nodo "edge" cumple funciones específicas de captura, procesamiento y transmisión. En el nivel local, los sensores conectados a cada nodo registran las señales físicas que luego son muestreadas mediante un control de frecuencia estable y un reloj sincronizado. Los datos adquiridos se almacenan temporalmente en un buffer circular (FIFO), que evita pérdidas por saturación, y posteriormente se organizan en ventanas de tiempo definidas para su análisis estadístico y extracción de características.

Cada nodo cuenta con una frecuencia de muestreo adaptada a su función: el nodo A opera a alta velocidad (100 Hz) y recopila el conjunto completo de sensores eléctricos, mecánicos y térmicos; el nodo B trabaja a 50 Hz con un subconjunto de sensores, mientras que el nodo C utiliza una frecuencia reducida (25 Hz) orientada a la robustez y estabilidad temporal. Esta estructura permite equilibrar el uso de recursos, minimizar la latencia y asegurar la fiabilidad de los datos incluso en condiciones de red variables. Asimismo, se definen indicadores clave para evaluar el rendimiento de cada nodo, tales como la frecuencia efectiva de muestreo, el jitter temporal, la pérdida de datos por desbordamiento del buffer y la disponibilidad del sistema. De esta manera, la arquitectura embebida propuesta constituye la base técnica que sustenta la integridad del proceso de adquisición y prepara los datos para su posterior transmisión mediante el protocolo MQTT y su integración en los modelos predictivos.

La Tabla 3 presenta el diccionario de variables utilizadas en el sistema de adquisición embebido, el cual describe los parámetros que intervienen en el proceso de captura, almacenamiento y procesamiento local de datos. Cada campo tiene una función específica orientada a garantizar la precisión temporal, la estabilidad de muestreo y la calidad de la información registrada por los nodos edge.

Tabla 3

Diccionario de variables del sistema de adquisición embebido

campo	definición
node_id	Identificador del nodo edge.
fs_objetivo_hz	Frecuencia de muestreo deseada del nodo.
fs_effective_hz	Frecuencia efectiva alcanzada (Hz).
max_jitter_ms	Jitter máximo (uniforme) aplicado a los instantes de muestreo.
sampling_jitter_ms_p95	Percentil 95 del jitter relativo (ms).
clock_drift_ppm	Deriva estática del reloj del nodo (ppm).
clock_wander_ms	Random walk de reloj por tick (ms).
buffer_capacity	Capacidad del buffer circular del nodo (muestras).
loss_overflow_pct	Porcentaje de muestras perdidas por overflow del buffer.
window_s	Duración de cada ventana de procesamiento local (s).
windows_out	Número total de ventanas generadas.
windows_rate_per_s	Ventanas por segundo.
coverage_s	Cobertura temporal adquirida (s).
dt_median_ms	Mediana del espaciado temporal entre muestras en la ventana (ms).
dt_p95_ms	Percentil 95 del espaciado temporal (ms).

Nota. Elaboración propia

La Tabla 4 muestra la configuración técnica de los nodos edge que conforman la arquitectura embebida distribuida para la adquisición de datos. Cada nodo fue diseñado con parámetros específicos de muestreo, almacenamiento y estabilidad temporal, adaptados a su función dentro del sistema.

Tabla 4

Configuración de Nodos Edge

node_id	fs_objetivo hz	Windows	Buffer capacity	Max jitter_ms	clock_drift ppm	clock_wander ms
edge-A	100	1	8000	0.2	3	0.05
edge-B	50	1	5000	0.6	8	0.1
edge-C	25	2	2500	1.5	20	0.2

Nota. Elaboración propia

El nodo edge-A opera con una frecuencia de muestreo de 100 Hz y una ventana de 1 segundo, lo que le permite captar señales de alta resolución, especialmente aquellas relacionadas con la vibración y la corriente eléctrica. Dispone de una capacidad de buffer de 8000 muestras, un jitter máximo de 0.2 ms y una deriva de reloj de solo 3 ppm, características

que garantizan una adquisición precisa y estable. El nodo edge-B, con una frecuencia de 50 Hz y las mismas condiciones de ventana, está orientado a la captura de variables de comportamiento térmico y mecánico. Su buffer de 5000 muestras y un jitter de 0.6 ms aseguran un equilibrio entre rendimiento y eficiencia energética, con una deriva de reloj moderada (8 ppm). Por último, el nodo edge-C trabaja a 25 Hz y con una ventana de 2 segundos, priorizando la estabilidad temporal y la reducción de carga de transmisión. Posee un buffer de 2500 muestras, un jitter máximo de 1.5 ms y una deriva de reloj de 20 ppm, valores adecuados para tareas de monitoreo de menor frecuencia.

La Tabla 5 presenta la configuración de sensores asignados a cada nodo edge dentro del sistema embebido distribuido. En ella se detallan los tipos de sensores utilizados, las unidades de medida y el nivel de ruido estándar (noise_std) asociado a cada señal, lo que permite evaluar la precisión y confiabilidad de las lecturas obtenidas.

Tabla 5

Configuración de sensores por nodo edge

node_id	sensor	unidad	noise_std
edge-A	motor_speed_rpm	rpm	0.5
edge-A	torque_Nm	Nm	0.8
edge-A	i_q_A	A	0.2
edge-A	i_d_A	A	0.1
edge-A	temp_winding_C	°C	0.05
edge-A	temp_bearing_C	°C	0.05
edge-A	ambient_C	°C	0.02
edge-B	motor_speed_rpm	rpm	0.7
edge-B	torque_Nm	Nm	1
edge-B	i_q_A	A	0.3
edge-B	i_d_A	A	0.15
edge-B	temp_winding_C	°C	0.05
edge-B	ambient_C	°C	0.02
edge-C	motor_speed_rpm	rpm	0.9
edge-C	torque_Nm	Nm	1.2
edge-C	i_q_A	A	0.35
edge-C	i_d_A	A	0.2
edge-C	temp_bearing_C	°C	0.05
edge-C	ambient_C	°C	0.02

Nota. Elaboración propia

El nodo edge-A concentra la mayor cantidad de sensores, ya que está destinado al monitoreo principal del motor. Registra variables críticas como velocidad (rpm), torque (Nm), corrientes en los ejes i_q e i_d (A), así como temperaturas del devanado, rodamiento y ambiente (°C). Este nodo posee niveles de ruido muy bajos (entre 0.02 y 0.8), lo que garantiza una adquisición precisa y estable para el análisis predictivo. El nodo edge-B se encarga de variables de comportamiento térmico y eléctrico intermedio, como la velocidad, torque, corrientes y temperatura del devanado, además de la temperatura ambiental. Los valores de ruido son ligeramente superiores a los del nodo principal, con un rango de 0.02 a 1, adecuados para operaciones de monitoreo secundario o redundante.

Por su parte, el nodo edge-C está orientado a tareas de soporte o respaldo, con sensores destinados al seguimiento de velocidad, torque, corrientes y temperatura de rodamientos, además de la temperatura ambiental. Aunque sus lecturas presentan un mayor nivel de ruido (hasta 1.2), mantiene una precisión suficiente para análisis de tendencias y detección de condiciones anómalas.

La Tabla 6 resume las métricas operativas de adquisición de datos obtenidas para cada nodo edge del sistema embebido distribuido. Estos indicadores permiten evaluar el rendimiento general del proceso de muestreo, la estabilidad temporal y la disponibilidad del sistema durante el periodo de observación.

Tabla 6

Métricas operativas de adquisición por nodo edge

Node id	samples	Los overflow pct	Fs effective hz	Sampling jitter ms p95	Coverage s	Node availability pct	Windows out	Windows rate per_s
edge-A	3000	0	100.001	0.322	29.99	100	30	1
edge-B	1500	0	50	0.922	29.98	100	30	1.001
edge-C	750	0	24.999	2.395	29.961	100	15	0.501

Nota. Elaboración propia

En el caso del nodo edge-A, se registraron 3000 muestras con una frecuencia efectiva de 100.001 Hz, lo que demuestra una sincronización precisa respecto a la frecuencia objetivo definida. El porcentaje de pérdida de datos (overflow) fue nulo y el jitter máximo registrado en el percentil 95 fue de 0.322 ms, lo que evidencia una excelente estabilidad temporal y una

cobertura de 29.99 segundos con una disponibilidad del 100 %. El nodo edge-B también mantuvo un desempeño estable, alcanzando una frecuencia efectiva de 50 Hz, sin pérdidas de datos y con un jitter p95 de 0.922 ms, dentro de los márgenes esperados para su configuración. Asimismo, generó 30 ventanas de procesamiento por segundo, manteniendo la misma cobertura temporal que el nodo principal. Por último, el nodo edge-C, que opera con menor frecuencia de muestreo (24.999 Hz), mostró un jitter p95 de 2.395 ms y una cobertura de 29.961 segundos, sin pérdida de muestras y con 100 % de disponibilidad. Si bien presenta una menor tasa de actualización (0.501 ventanas por segundo), su desempeño es consistente con su función de monitoreo complementario.

4.3 Transmisión y sincronización de datos mediante el protocolo MQTT

La presente sección aborda el proceso de transmisión y sincronización de datos entre los nodos de adquisición y el servidor principal mediante el protocolo MQTT (Message Queuing Telemetry Transport). Este protocolo se seleccionó por su eficiencia en entornos embebidos, su bajo consumo de ancho de banda y su capacidad para mantener la integridad de la información incluso en redes con latencias variables. El sistema de comunicación MQTT implementa una estructura cliente-servidor (publicador-suscriptor), donde cada nodo edge actúa como publicador enviando los paquetes de datos adquiridos hacia un broker central encargado de su recepción, ordenamiento y distribución. Cada mensaje transmitido contiene un identificador único, marca temporal, carga útil (payload), nivel de calidad de servicio (QoS) y métricas asociadas a la latencia y ancho de banda utilizado.

La Tabla 7 muestra la trazabilidad de los principales parámetros de transmisión para los tres nodos (edge-A, edge-B y edge-C). Se observa que las latencias promedio se mantienen dentro de márgenes aceptables, variando entre 35 y 55 ms para el nodo edge-A, 60 a 110 ms para el edge-B, y hasta 200 ms para el edge-C, debido a su menor frecuencia de envío. En todos los casos, el campo lost registra valor FALSO, lo que confirma la ausencia de pérdida de paquetes durante la transmisión.

En cuanto al tamaño de la carga útil, los mensajes presentan variaciones entre 850 y 1600 bytes, dependiendo del volumen de datos de cada ventana enviada, mientras que el ancho de banda empleado se mantiene dentro de rangos de 130 a 320 kbps en los nodos de mayor frecuencia. El nivel de servicio QoS utilizado (1 o 2) garantiza la entrega segura de los mensajes, priorizando la confiabilidad sobre la velocidad.

Tabla 7

Trazabilidad de latencia, pérdida y carga útil por nodo MQTT

node_id	Msg id	Timestamps	latency_ms	lost	Payload bytes	Bandwidth kbps	qos
edge-A	0	0	47.4377366	FALSO	1146	193.2638581	2
edge-A	1	1	52.5245177	FALSO	1486	226.332397	2
edge-A	2	2	34.5825639	FALSO	1373	317.6167047	2
edge-A	3	3	44.8655907	FALSO	1428	254.6272057	2
edge-A	4	4	52.0351838	FALSO	1425	219.0825355	2
edge-A	5	5	54.0179297	FALSO	1315	194.7501518	2
edge-A	6	6	38.1256603	FALSO	1510	316.8469715	2
edge-A	7	7	52.0276024	FALSO	851	130.8536178	2
edge-A	8	8	43.5211011	FALSO	854	156.9813224	2
edge-A	9	9	43.7637641	FALSO	1576	288.0922207	2
edge-A	10	10	42.1829316	FALSO	1198	227.2008994	2
edge-A	11	11	48.3018609	FALSO	835	138.2969492	2
edge-A	12	12	62.1331808	FALSO	1128	145.236408	2
edge-A	13	13	38.4898182	FALSO	1060	220.3180062	2
edge-A	14	14	54.0317783	FALSO	1349	199.7343106	2
edge-A	15	15	38.4041503	FALSO	1180	245.8067665	2
edge-A	16	16	50.9460334	FALSO	1303	204.6086675	2
edge-A	17	17	49.3452341	FALSO	1360	220.4873518	2
edge-A	18	18	46.8572906	FALSO	1518	259.1699146	2
edge-A	19	19	51.9714302	FALSO	1030	158.5486481	2
edge-A	20	20	50.4313085	FALSO	1437	227.9536332	2
edge-A	21	21	50.0503058	FALSO	1429	228.4101928	2
edge-A	22	22	42.4426303	FALSO	1204	226.9416372	2
edge-A	23	23	42.798862	FALSO	1254	234.3987558	2
edge-A	24	24	38.0733511	FALSO	1484	311.8191508	2
edge-A	25	25	42.3209198	FALSO	1252	236.6678243	2
edge-A	26	26	49.6897787	FALSO	1435	231.03343	2
edge-A	27	27	42.2101994	FALSO	1043	197.6773414	2
edge-A	28	28	51.8638071	FALSO	1594	245.8747386	2
edge-A	29	29	35.9337023	FALSO	1482	329.9409536	2
edge-B	0	0	89.9432149	FALSO	1147	102.0199246	1
edge-B	1	1	71.4549471	FALSO	1329	148.7930568	1
edge-B	2	2	92.5118079	FALSO	1134	98.06315764	1
edge-B	3	3	66.7614812	FALSO	1451	173.8727152	1
edge-B	4	4	72.3652421	FALSO	1357	150.0167716	1
edge-B	5	5	70.6119532	FALSO	1169	132.4421656	1
edge-B	6	6	89.6149332	FALSO	1196	106.7679198	1
edge-B	7	7	78.0302903	FALSO	1156	118.5180776	1
edge-B	8	8	78.4056358	FALSO	1567	159.8864657	1

node_id	Msg id	Timestamps	latency_ms	lost	Payload bytes	Bandwidth kbps	qos
edge-B	9	9	53.5460078	FALSO	870	129.9816792	1
edge-B	10	10	87.9954845	FALSO	1165	105.9145256	1
edge-B	11	11	105.984566	FALSO	1012	76.38848097	1
edge-B	12	12	94.7503114	FALSO	1389	117.2766595	1
edge-B	13	13	60.9995589	FALSO	1017	133.3780137	1
edge-B	14	14	96.8061627	FALSO	1377	113.7944082	1
edge-B	15	15	84.754712	FALSO	1044	98.54319363	1
edge-B	16	16	51.0788429	FALSO	1194	187.0050192	1
edge-B	17	17	84.6535242	FALSO	1375	129.9414301	1
edge-B	18	18	112.035578	FALSO	1302	92.97046708	1
edge-B	19	19	83.5855127	FALSO	867	82.98088719	1
edge-B	20	20	107.183752	FALSO	1328	99.11950133	1
edge-B	21	21	109.266058	FALSO	915	66.99244162	1
edge-B	22	22	67.2049693	FALSO	1591	189.3907567	1
edge-B	23	23	52.466277	FALSO	1264	192.7333245	1
edge-B	24	24	75.5555461	FALSO	1134	120.0706033	1
edge-B	25	25	86.2702769	FALSO	1185	109.8872095	1
edge-B	26	26	119.934618	FALSO	1152	76.8418674	1
edge-B	27	27	60.2092376	FALSO	1550	205.948464	1
edge-B	28	28	85.3542292	FALSO	1236	115.8466322	1
edge-B	29	29	67.758064	FALSO	1216	143.5696273	1
edge-C	0	0	162.639209	FALSO	1234	60.69877028	0
edge-C	1	2	78.5738499	FALSO	912	92.85532032	0
edge-C	2	4	100.547684	FALSO	995	79.16641848	0
edge-C	3	6	190.717197	FALSO	1327	55.6635699	0
edge-C	4	8	159.30958	FALSO	1580	79.34237203	0
edge-C	5	10	81.3953295	FALSO	984	96.71316582	0
edge-C	6	12	205.138789	FALSO	1168	45.54964974	0
edge-C	7	14	83.8829129	FALSO	1446	137.9065128	0
edge-C	8	16	135.398039	FALSO	1524	90.04561751	0
edge-C	9	18	93.8084922	FALSO	1004	85.62124615	0
edge-C	10	20	101.800661	FALSO	1575	123.7712988	0
edge-C	11	22	126.903517	FALSO	1593	100.422749	0
edge-C	12	24	126.399665	FALSO	892	56.45584603	0
edge-C	13	26	172.24007	FALSO	1215	56.43286152	0
edge-C	14	28	103.562911	FALSO	1509	116.5668279	0

Nota. Elaboración propia

La Tabla 8 presenta las métricas de rendimiento del enlace MQTT obtenidas para cada nodo edge, con el propósito de evaluar la eficiencia, estabilidad y confiabilidad del canal de comunicación entre los dispositivos embebidos y el servidor central.

Tabla 8

Métricas de rendimiento de enlace MQTT por nodo edge

node_id	Los pct	Throughput msg_s	Latency mean_ms	Latency p95_ms	Bandwidth mean_kbps	Availability pct	qos
edge-A	0	1	46.38	54.03	226.29	100	2
edge-B	0	1	81.9	110.79	125.97	100	1
edge-C	0	0.5	128.15	195.04	85.15	93.33	0

Nota. Elaboración propia

El nodo edge-A, configurado con un nivel de servicio QoS 2, muestra el mejor desempeño general. No presenta pérdidas de mensajes (loss pct = 0) y mantiene un throughput de 1 mensaje por segundo, con una latencia promedio de 46.38 ms y un percentil 95 (p95) de 54.03 ms. Además, su ancho de banda medio alcanza 226.29 kbps y una disponibilidad del 100 %, evidenciando una comunicación estable y de alta fiabilidad, ideal para el envío de datos críticos. El nodo edge-B, que opera con QoS 1, también mantiene una transmisión sin pérdidas y una tasa de envío constante de 1 mensaje por segundo, aunque con una latencia promedio de 81.9 ms y un p95 de 110.79 ms, producto de su menor prioridad de servicio. Su ancho de banda medio es de 125.97 kbps, suficiente para garantizar la correcta entrega de los datos adquiridos.

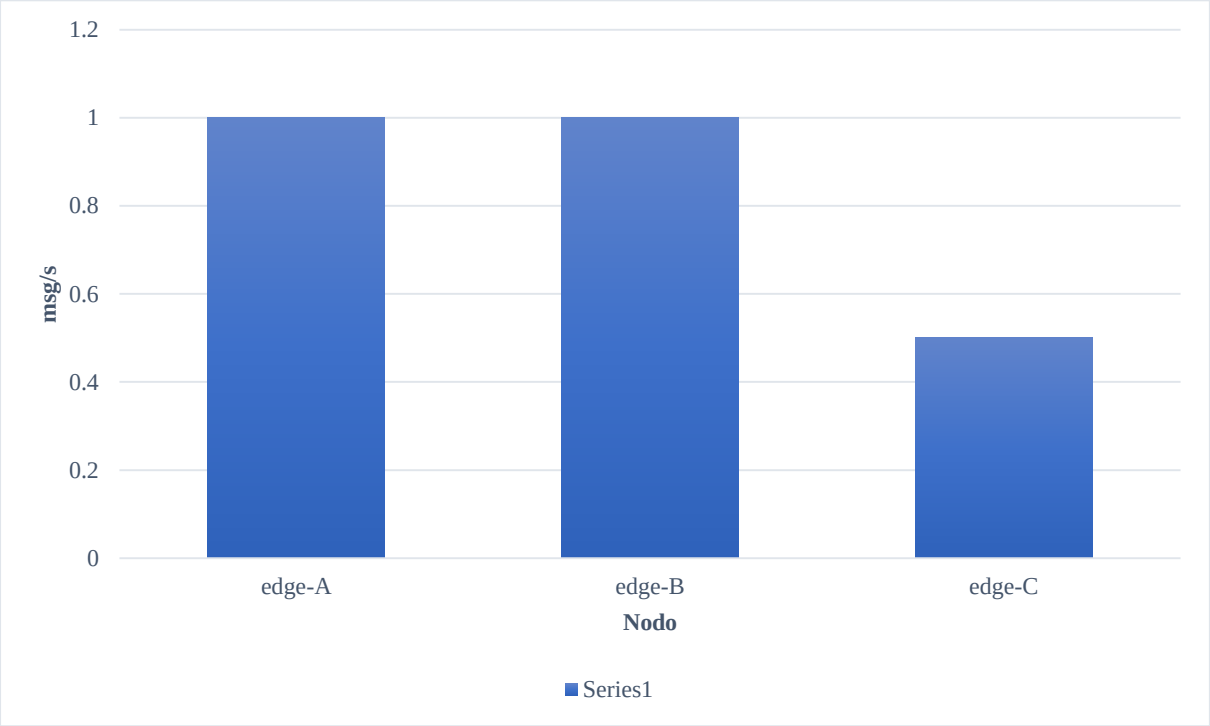
Por su parte, el nodo edge-C, configurado con QoS 0, presenta un throughput de 0.5 mensajes por segundo y una latencia más elevada (promedio de 128.15 ms, p95 de 195.04 ms), junto con un ancho de banda medio de 85.15 kbps y una disponibilidad del 93.33 %. Si bien su rendimiento es inferior, resulta aceptable considerando su función secundaria y su menor frecuencia de muestreo.

La Figura 14 ilustra el throughput promedio (mensajes por segundo) alcanzado por cada nodo edge durante la transmisión de datos mediante el protocolo MQTT. Este indicador permite comparar la capacidad de envío sostenido de mensajes entre los distintos nodos del sistema embebido distribuido. Se observa que los nodos edge-A y edge-B mantienen un throughput de 1 mensaje por segundo, lo que demuestra una transmisión continua, estable y sin interrupciones, acorde con los parámetros configurados en el enlace MQTT. Este comportamiento confirma una sincronización adecuada y un flujo de datos constante hacia el

servidor central. En contraste, el nodo edge-C registra un throughput de 0.5 mensajes por segundo, resultado esperado dada su menor frecuencia de muestreo y su configuración con QoS 0, orientada a reducir la carga de red y el consumo energético.

Figura 14

Throughput (msg/s) por nodo



Nota. Elaboración propia

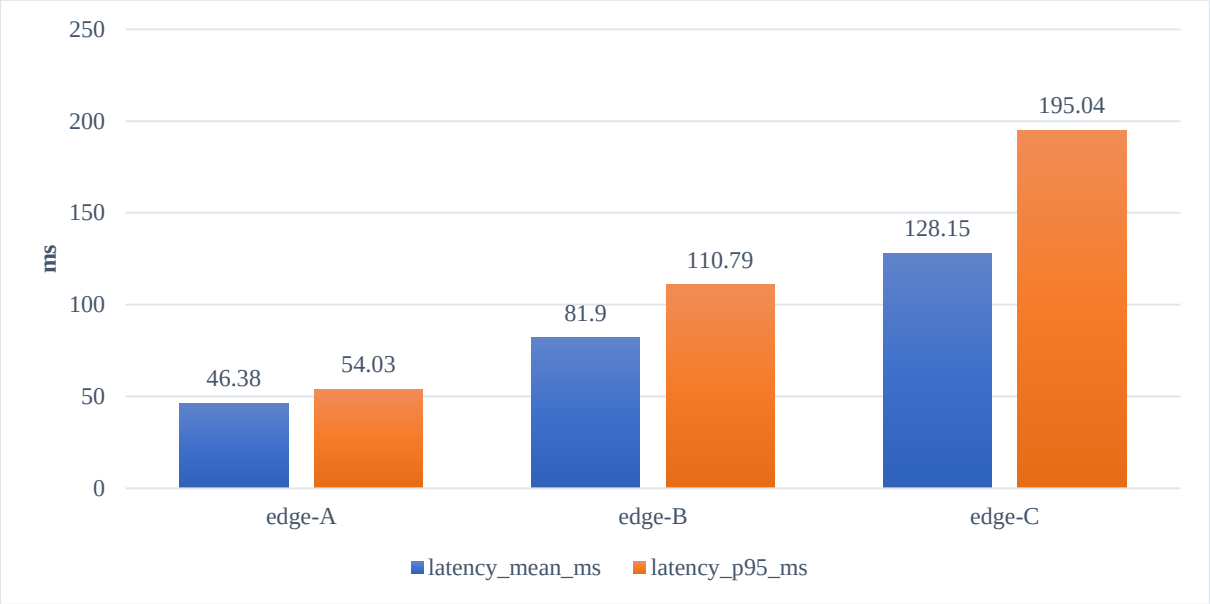
La Figura 15 muestra la comparación de la latencia promedio y la latencia en el percentil 95 (p95) para cada nodo edge del sistema de transmisión MQTT. Este análisis permite evaluar la rapidez y estabilidad del envío de datos desde los dispositivos embebidos hacia el servidor central. El nodo edge-A presenta los valores más bajos de latencia, con una media de 46.38 ms y un p95 de 54.03 ms, lo que evidencia una comunicación rápida y constante, acorde con su configuración de QoS 2, que prioriza la entrega garantizada de los mensajes. El nodo edge-B muestra una latencia promedio de 81.9 ms y un p95 de 110.79 ms, incrementos esperados por su menor prioridad de servicio (QoS 1) y un flujo de transmisión ligeramente más variable.

Por otro lado, el nodo edge-C registra los valores más altos, con una media de 128.15 ms y un p95 de 195.04 ms, producto de su configuración con QoS 0 y su menor frecuencia de

envío de datos. Aun así, los valores permanecen dentro de márgenes aceptables para aplicaciones de monitoreo en tiempo casi real.

Figura 15

Latencia (ms) media y p95 por nodo



Nota. Elaboración propia

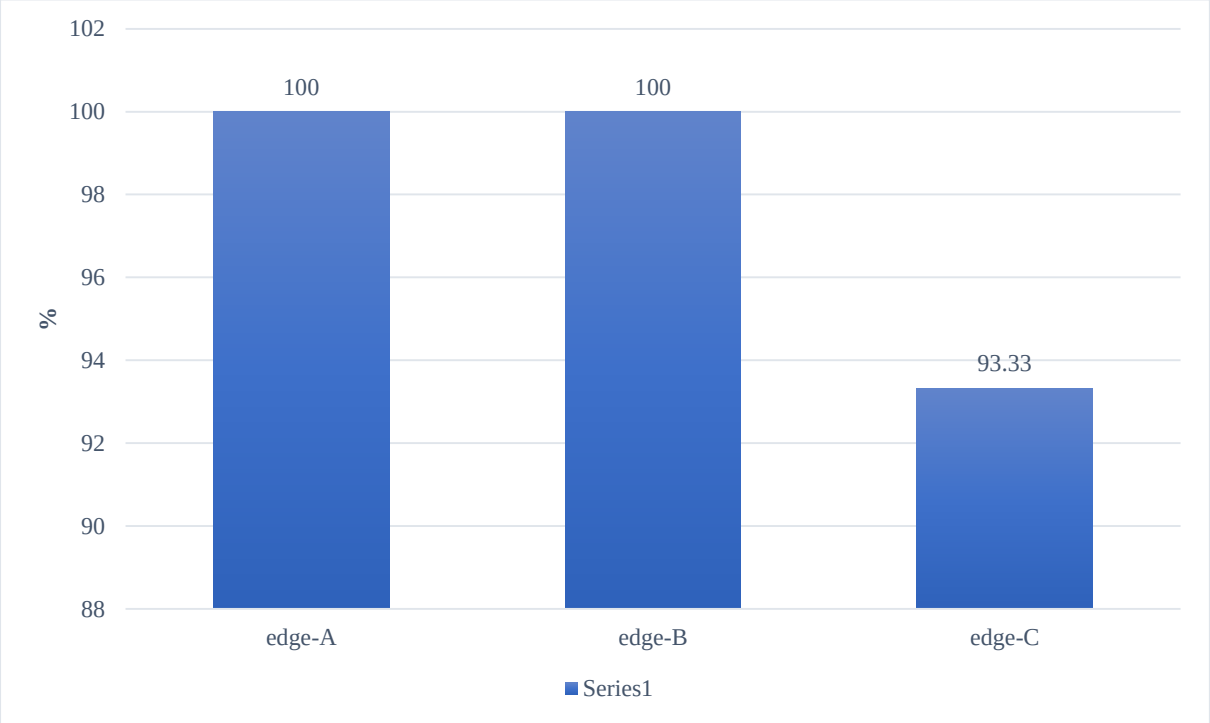
La Figura 16 muestra el nivel de disponibilidad (%) alcanzado por cada nodo edge durante la transmisión de datos mediante el protocolo MQTT. Este indicador refleja la proporción de tiempo en que cada nodo mantuvo una conexión activa y funcional con el servidor central, garantizando la entrega continua de información.

Los resultados evidencian una disponibilidad del 100 % en los nodos edge-A y edge-B, lo que indica un funcionamiento estable y sin interrupciones durante todo el periodo de monitoreo. Esto se asocia al uso de niveles de servicio QoS 2 y QoS 1, respectivamente, los cuales priorizan la confiabilidad del enlace y aseguran la confirmación de entrega de los mensajes.

Por su parte, el nodo edge-C alcanzó una disponibilidad del 93.33 %, valor ligeramente inferior debido a su configuración con QoS 0 y su menor frecuencia de transmisión. Aunque presenta pequeñas interrupciones, su rendimiento se mantiene dentro de márgenes aceptables para aplicaciones de respaldo o monitoreo complementario.

Figura 16

Disponibilidad (%) por nodo



Nota. Elaboración propia

La Tabla 9 presenta los resultados del proceso de sincronización temporal entre los nodos edge del sistema de transmisión MQTT. Este procedimiento tiene como objetivo mantener la coherencia de los sellos de tiempo (timestamps) generados por cada nodo durante la adquisición y envío de datos, corrigiendo las desviaciones que se producen por deriva de reloj (clock drift).

Tabla 9

Resultados de sincronización temporal entre nodos MQTT

node_id	sync_cycle	drift_before_ms	correction_applied_ms
edge-A	1	4.607267976	-4.607267976
edge-A	2	5.156971288	-2.578485644
edge-A	3	1.483564192	-0.494521397
edge-A	4	-2.620913405	0.655228351
edge-A	5	2.331870391	-0.466374078
edge-A	6	7.502867554	-1.250477926
edge-A	7	6.964309914	-0.994901416
edge-A	8	5.814747951	-0.726843494
edge-A	9	10.19908083	-1.133231203

node_id	sync_cycle	drift_before_ms	correction_applied_ms
edge-A	10	6.877943781	-0.687794378
edge-B	1	-2.684181057	2.684181057
edge-B	2	-0.754200673	0.377100336
edge-B	3	-1.938016041	0.646005347
edge-B	4	-1.953381642	0.48834541
edge-B	5	-2.443710337	0.488742067
edge-B	6	-1.430986691	0.238497782
edge-B	7	2.791458893	-0.398779842
edge-B	8	3.063213614	-0.382901702
edge-B	9	4.995029994	-0.555003333
edge-B	10	-1.155486309	0.115548631
edge-C	1	-0.146155206	0.146155206
edge-C	2	-2.675846017	1.337923008
edge-C	3	-6.332285198	2.110761733
edge-C	4	-8.966742299	2.241685575
edge-C	5	-9.969112621	1.993822524
edge-C	6	-7.221404994	1.203567499
edge-C	7	-11.20058315	1.600083307
edge-C	8	-11.10868867	1.388586084
edge-C	9	-12.56119697	1.395688552
edge-C	10	-13.54421625	1.354421625

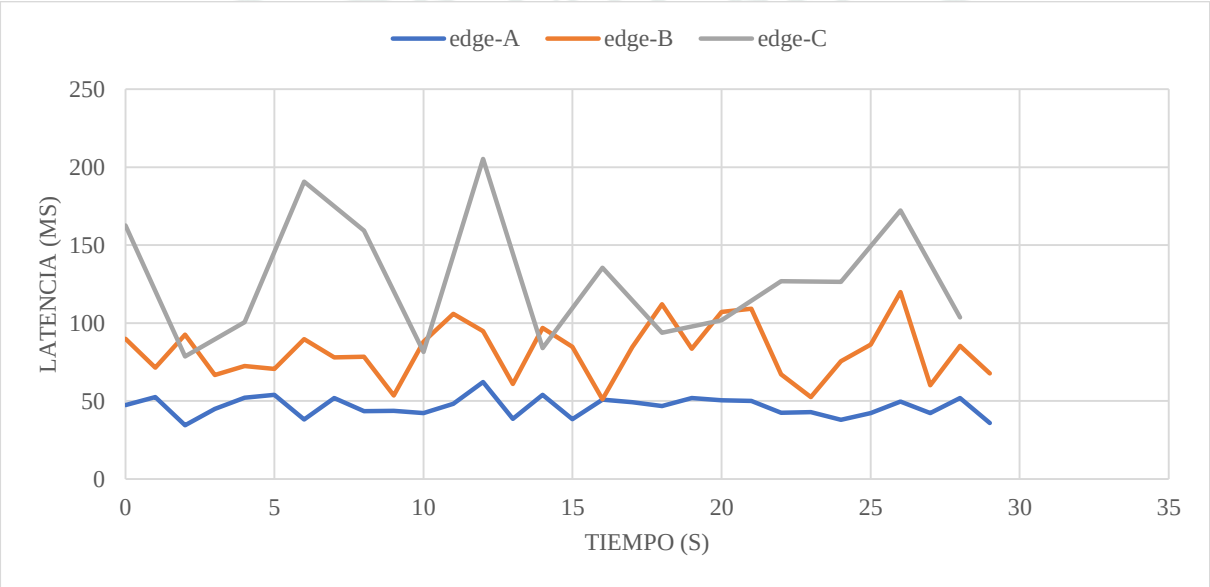
Nota. Elaboración propia

El parámetro sync_cycle representa el número de ciclo de sincronización ejecutado, mientras que drift_before_ms indica la desviación temporal acumulada antes de aplicar la corrección, expresada en milisegundos. Por su parte, correction_applied_ms muestra el valor ajustado por el algoritmo de sincronización para compensar dicho error, aplicando una fracción proporcional que evita saltos bruscos en la señal temporal. En el caso del nodo edge-A, las desviaciones previas a la corrección oscilan entre 1.4 y 10.2 ms, evidenciando pequeñas fluctuaciones del reloj local. Las correcciones aplicadas son menores a $\pm$1.5 ms en la mayoría de los ciclos, lo que demuestra una alta precisión en el ajuste. El nodo edge-B presenta valores de deriva moderada, con variaciones entre -2.6 y 5 ms, compensadas mediante correcciones de baja magnitud (±0.6 ms), manteniendo una sincronización estable y predecible. En cambio, el nodo edge-C muestra las mayores desviaciones negativas (hasta -13.5 ms), propias de un reloj con mayor deriva y menor estabilidad. Aun así, el sistema logra compensarlas parcialmente mediante correcciones de entre 1 y 2.2 ms, mejorando su alineación con los demás nodos.

La Figura 17 muestra la serie temporal de latencias registradas durante la transmisión de datos de los tres nodos edge a través del protocolo MQTT. Este gráfico permite visualizar la estabilidad del enlace y las diferencias en el comportamiento de cada nodo en función de su configuración de calidad de servicio (QoS) y frecuencia de envío.

Figura 17

Serie de latencias



Nota. Elaboración propia

El nodo edge-A (línea azul) presenta los valores más bajos y estables, con latencias que oscilan entre 40 y 60 ms, evidenciando una comunicación fluida y confiable gracias a su configuración con QoS 2, que asegura la entrega confirmada de los mensajes. El nodo edge-B (línea naranja) mantiene latencias intermedias, fluctuando entre 70 y 120 ms, con algunos picos puntuales producto de variaciones en la red o tiempos de procesamiento. Este comportamiento es coherente con su configuración de QoS 1, que prioriza la eficiencia sin requerir confirmación doble. En contraste, el nodo edge-C (línea gris) exhibe una latencia más variable y elevada, con valores que llegan hasta 200 ms. Esta irregularidad se asocia a su QoS 0 y menor tasa de transmisión, que no garantizan confirmación de entrega ni priorización del canal.

CAPÍTULO V

RESULTADOS Y VALIDACIÓN DEL SISTEMA PREDICTIVO PROPUESTO

5.1 Modelos predictivos

En esta sección se presentan los modelos predictivos desarrollados para analizar el comportamiento operativo del motor eléctrico y anticipar posibles fallas a partir de los datos adquiridos por el sistema embebido. Estos modelos se construyen utilizando las variables procesadas durante la etapa de adquisición como vibración, temperatura y corriente y emplean técnicas de aprendizaje supervisado orientadas a la estimación del estado de degradación del equipo. El propósito de esta etapa es establecer una base analítica que permita predecir tendencias anómalas y patrones de deterioro, contribuyendo a la implementación de estrategias de mantenimiento predictivo más precisas y eficientes. Para ello, se evalúan distintos enfoques de modelado y se comparan métricas de desempeño que determinan su capacidad de generalización y confiabilidad dentro del entorno operativo monitoreado.

5.1.1 XGBoost por escenarios

Para la evaluación predictiva del comportamiento del motor eléctrico se utilizó el modelo XGBoost (Extreme Gradient Boosting), una técnica de aprendizaje supervisado basada en árboles de decisión que permite manejar relaciones no lineales entre variables y ofrecer alta precisión en la predicción de fallas. Este modelo fue seleccionado por su capacidad de optimización iterativa y su buen desempeño en problemas de regresión y clasificación con datos multivariables.

Con el fin de analizar la respuesta del sistema bajo diferentes condiciones operativas, se definieron tres escenarios de simulación:

- **Escenario Base:** Representa la tendencia progresiva de falla del motor bajo condiciones normales de operación.
- **Escenario Conservador:** Considera un entorno de operación controlado, con variaciones limitadas en las variables térmicas y mecánicas.
- **Escenario Estresado:** Simula condiciones severas de operación, como incrementos en temperatura, vibración y corriente por encima de los rangos normales.

Tabla 10

Escenario Base – Tendencia progresiva de falla del motor

vib_mm_s rms	Acc pico_g	Temp devanadoC	Temp rodamiento C	ambiente C	Corriente A	Corriente pctFLA	month	prob_mean	prob_p10	prob_p90
1.73	0.085	77.9	37	31	100.6	77.4	0	0.09218	0.02263	0.16808
1.810412093	0.08422	78.9846	37.5033	31.0609	102.009	77.5545	1	0.16183	0.03847	0.32429
1.888831077	0.08456	79.5774	37.8322	31.0771	102.337	77.7388	2	0.19322	0.03847	0.40134
1.965600212	0.08473	80.2623	38.3545	31.1064	102.947	77.8742	3	0.23825	0.04799	0.55458
1.998301228	0.08483	81.444	38.6544	31.0725	103.953	77.9022	4	0.40742	0.09856	0.82613
2.115832733	0.08525	82.2344	39.5873	31.093	104.656	78.007	5	0.42776	0.09856	0.8842
2.31082683	0.08514	82.5421	40.5195	31.087	104.587	78.1392	6	0.641	0.30123	0.93805
2.38406042	0.08506	83.2417	41.1766	31.1225	105.213	78.2194	7	0.84658	0.68871	0.9798
2.471083087	0.08569	84.0263	41.496	31.1768	105.798	78.3458	8	0.91857	0.87699	0.98507
2.589447452	0.08726	84.5592	41.9912	31.1758	106.477	78.4711	9	0.9595	0.93944	0.99062
2.608631922	0.08634	84.8389	42.8763	31.2613	107.235	78.4446	10	0.96107	0.93852	0.99204
2.71213453	0.08735	85.1607	43.5478	31.3101	108.236	78.4744	11	0.98806	0.98504	0.99522
2.862186835	0.08498	85.9641	43.9516	31.2186	109.24	78.69	12	0.98766	0.97595	0.99564
2.955314409	0.08579	87.0477	44.946	31.3006	110.337	78.7019	13	0.98811	0.97605	0.99575
2.975216118	0.08558	87.3948	46.1973	31.2918	111.186	78.7887	14	0.98817	0.97675	0.99575
3.077396241	0.08703	87.808	46.8557	31.2546	111.458	78.6923	15	0.99143	0.98858	0.99629
3.175544401	0.08666	88.2002	47.0338	31.2505	111.656	78.8119	16	0.98979	0.97675	0.9963
3.279666388	0.0854	88.7075	47.4082	31.2664	112.993	78.5163	17	0.98952	0.97675	0.9963
3.385417692	0.08526	89.5089	48.0721	31.2288	113.665	78.3913	18	0.98939	0.97657	0.99611
3.539218049	0.08528	90.6792	48.6539	31.1934	113.86	78.1086	19	0.99151	0.98785	0.99658
3.560011252	0.08551	91.4319	48.7796	31.2382	114.466	78.1315	20	0.99169	0.98785	0.99658
3.740622284	0.08627	91.9513	49.1122	31.1306	114.979	78.0985	21	0.99188	0.98785	0.99681
3.791263307	0.08616	92.5744	49.6447	31.0981	115.846	78.1676	22	0.99205	0.98785	0.99681
3.836743353	0.08738	93.1448	50.2504	31.1335	116.739	78.3166	23	0.99383	0.99081	0.9975

vib_mm_s rms	Acc pico_g	Temp devanadoC	Temp rodamiento C	ambiente C	Corriente A	Corriente pctFLA	month	prob_mean	prob_p10	prob_p90
3.951099562	0.08684	93.8532	50.5073	31.1389	117.526	78.284	24	0.99232	0.98819	0.99691
4.044672686	0.08521	94.2886	51.1351	31.1469	117.915	78.4822	25	0.99223	0.98819	0.9967
4.101363944	0.08603	95.0168	51.5413	31.2357	118.238	78.6264	26	0.99252	0.98829	0.99693
4.251393059	0.08545	96.0345	52.0663	31.2501	119.142	78.6053	27	0.99233	0.98829	0.99689
4.391933473	0.08613	96.11	53.0576	31.167	120.355	78.7824	28	0.99252	0.98829	0.99693
4.437385152	0.08578	96.3782	53.5931	31.1516	121.467	78.9889	29	0.99233	0.98829	0.99689
4.607169342	0.0853	96.9155	53.688	31.1113	122.653	79.2441	30	0.99236	0.98829	0.9969
4.645451899	0.08479	97.5076	54.3721	31.0708	123.622	79.169	31	0.99235	0.98829	0.9969
4.744729285	0.08525	98.1095	54.8412	31.0653	124.387	79.1104	32	0.99238	0.98829	0.9969
4.831789482	0.08662	98.8778	55.3525	31.1002	125.053	79.296	33	0.99256	0.98829	0.99694
4.950271592	0.08652	99.6121	55.4813	31.0438	125.376	79.3187	34	0.99274	0.98829	0.99742
5.136897341	0.08846	100.403	56.2315	30.9538	125.959	79.1937	35	0.99555	0.99332	0.99779
5.205854962	0.08831	100.862	56.2377	30.9912	126.33	79.2316	36	0.99555	0.99332	0.99779

Nota. Elaboración propia

La Tabla 10 muestra el comportamiento del escenario base, que representa la tendencia progresiva de falla del motor eléctrico bajo condiciones normales de operación. En este escenario se observa una evolución gradual de los parámetros operativos vibración, aceleración pico, temperatura del devanado, temperatura del rodamiento, temperatura ambiente y corriente a lo largo del tiempo (variable month), reflejando un patrón de desgaste acumulativo en el sistema.

A medida que avanza el ciclo de operación, se aprecia un incremento sostenido en la vibración RMS (de 1.73 a 5.20 mm/s) y en las temperaturas del devanado y rodamiento, lo que indica un aumento de fricción interna y sobrecalentamiento progresivo. Del mismo modo, la corriente total y la corriente porcentual respecto al FLA muestran una ligera tendencia ascendente, reflejando un mayor esfuerzo eléctrico del motor conforme se aproxima al estado de falla. En cuanto a las predicciones del modelo XGBoost, los valores de probabilidad media (prob_mean) se incrementan desde 0.09 hasta valores cercanos a 0.99, confirmando la tendencia al deterioro operativo. Los percentiles prob_p10 y prob_p90 muestran una estrecha convergencia en las últimas etapas, lo que indica alta confianza del modelo en la predicción de la falla.

La Tabla 11 presenta los resultados correspondientes al escenario conservador, diseñado para evaluar el desempeño del modelo XGBoost bajo condiciones de operación controladas y estables. En este caso, las variaciones de temperatura, vibración y corriente se mantienen dentro de límites estrechos, simulando un entorno de funcionamiento preventivo con baja exigencia térmica y mecánica.

Se observa que los valores de vibración RMS (vib_mm_s_rms) y aceleración pico (acc_pico_g) crecen de forma moderada, al igual que las temperaturas del devanado y rodamiento, que permanecen en rangos operativos seguros. La corriente eléctrica (A) y el porcentaje de carga (%FLA) presentan incrementos leves, indicando una progresión lenta en el esfuerzo del motor. Desde el punto de vista predictivo, los valores de prob_mean, prob_p10 y prob_p90 muestran una tendencia ascendente más gradual en comparación con el escenario base. Esto significa que el modelo identifica un aumento paulatino en la probabilidad de falla, pero sin llegar a niveles críticos en los primeros ciclos de operación.

Tabla 11

Escenario Conservador – Tendencia progresiva de falla del motor

vib_mm_s_rms	acc_pico_g	temp_devanado_C	temp_rodamiento_C	ambiente_C	corriente_A	corriente_pctFLA	month	prob_mean	prob_p10	prob_p90
1.73	0.085	77.9	37	31	100.6	77.4	0	0.09218	0.02263	0.16808
1.84222	0.08422	77.6939	37.2428	31.055	101.292	77.2097	1	0.12799	0.02938	0.29182
1.90712	0.08383	77.9728	38.0717	30.9799	101.642	77.2819	2	0.18285	0.03413	0.419
1.9897	0.08391	78.3344	38.2933	30.9934	101.874	77.378	3	0.23152	0.04336	0.55264
2.02861	0.08423	78.6285	38.8185	30.9908	101.819	77.5732	4	0.25004	0.04528	0.62261
2.05144	0.08367	78.8094	39.3934	30.884	102.747	77.7371	5	0.29827	0.06185	0.69476
2.10813	0.08363	79.1665	39.7977	30.9163	103.158	77.751	6	0.3796	0.07363	0.87865
2.15221	0.0835	79.4275	40.3244	30.9459	102.87	77.853	7	0.36496	0.06742	0.76674
2.20806	0.08349	79.6498	40.8492	30.9933	103.015	77.9164	8	0.41841	0.09993	0.82946
2.22891	0.08233	79.8245	41.3217	31.041	103.418	78.0294	9	0.50822	0.12931	0.92797
2.29439	0.08181	80.2108	41.7818	31.0787	103.956	78.1399	10	0.62878	0.29923	0.93396
2.35493	0.08268	80.5055	41.9054	31.0705	104.107	78.0344	11	0.8447	0.68871	0.9798
2.4006	0.08203	80.7439	41.8267	31.0812	104.407	77.9069	12	0.84493	0.68871	0.9798
2.47038	0.0828	80.7224	41.9742	31.0427	104.519	78.0361	13	0.90575	0.82479	0.98476
2.51456	0.08258	80.6217	42.4012	30.9808	104.831	78.0794	14	0.9068	0.83779	0.98476
2.60538	0.0815	80.3178	42.738	31.0708	105.379	78.1122	15	0.9484	0.88726	0.98811
2.64063	0.08215	80.4231	43.1333	31.0961	105.891	78.5433	16	0.97716	0.95671	0.99277
2.70241	0.08174	80.7179	43.0795	31.0613	106.109	78.461	17	0.97831	0.96545	0.99249
2.75698	0.08106	81.2421	43.3795	31.0609	106.343	78.4853	18	0.98211	0.96633	0.99328
2.83173	0.08036	81.4105	43.6188	30.9936	106.479	78.4106	19	0.98223	0.96672	0.99332
2.90796	0.08057	81.7492	44.0889	30.9867	106.857	78.3944	20	0.98547	0.9707	0.99483
2.96592	0.0801	81.5615	44.3621	31.0579	107.343	78.5457	21	0.98609	0.97088	0.99507
3.05301	0.08017	81.9009	44.5386	31.042	107.766	78.7349	22	0.98721	0.97088	0.99553
3.09882	0.07981	82.201	44.9588	30.9699	107.607	78.6482	23	0.98726	0.97088	0.99565

vib_mm_s_rms	acc_pico_g	temp_devanado_C	temp_rodamiento_C	ambiente_C	corriente_A	corriente_pctFLA	month	prob_mean	prob_p10	prob_p90
3.1122	0.07941	82.4715	45.1681	31.0425	107.972	78.6954	24	0.98835	0.97088	0.99553
3.18087	0.0785	82.9793	45.453	31.076	108.058	78.4509	25	0.98804	0.97057	0.99537
3.2554	0.0788	83.0989	45.5791	31.1323	108.12	78.7086	26	0.98885	0.97077	0.99585
3.27838	0.0793	83.3332	45.9997	31.1051	108.387	78.7151	27	0.98898	0.97088	0.99585
3.29832	0.07989	83.5305	46.254	31.0554	108.175	78.685	28	0.98898	0.97088	0.99585
3.4207	0.08052	83.8267	46.5014	31.0565	108.584	78.4992	29	0.98908	0.9754	0.99582
3.50413	0.08109	84.2703	46.8891	31.0575	109.019	78.5921	30	0.98927	0.97558	0.9962
3.54351	0.08063	84.5907	47.4989	30.9955	109.025	78.618	31	0.98928	0.97558	0.9962
3.59503	0.08077	84.9395	47.7543	31.07	109.005	78.4877	32	0.98911	0.9754	0.99617
3.64193	0.07944	85.2043	48.3866	31.071	109.351	78.2899	33	0.98905	0.97587	0.99589
3.67344	0.07914	85.4408	48.9429	31.0989	109.429	78.3285	34	0.98905	0.97587	0.99589
3.73839	0.08004	86.0569	49.0399	31.0584	109.477	78.2918	35	0.98905	0.97587	0.99589
3.80238	0.08083	86.3417	49.2759	31.066	109.61	78.619	36	0.98934	0.97605	0.9963

Nota. Elaboración propia

En el escenario conservador de la Tabla 11, los resultados muestran una evolución más estable del comportamiento del motor, con incrementos moderados en las variables térmicas, mecánicas y eléctricas. Este escenario representa una condición operativa donde el sistema trabaja bajo un entorno controlado, con niveles reducidos de vibración y fluctuaciones suaves de temperatura y corriente. Sin embargo, a pesar de esta estabilidad inicial, el modelo XGBoost identifica un incremento sostenido en la probabilidad de falla (prob_mean) conforme avanza el tiempo. En los primeros meses (de 0 a 5), la probabilidad de falla se mantiene baja, por debajo de 0.3, lo que indica un funcionamiento normal del motor. Entre los meses 6 y 10, se observa un crecimiento progresivo, alcanzando valores entre 0.4 y 0.6, reflejando el inicio de un desgaste incipiente en el sistema. A partir del mes 12, la probabilidad de falla supera el 0.8, y hacia los meses 15 a 17, los valores de prob_mean y prob_p90 se aproximan a 0.98-0.99, lo que evidencia un alto riesgo de falla inminente. En este punto, el motor presenta condiciones térmicas elevadas (temperatura del devanado cercana a 80 °C y del rodamiento sobre 43 °C) y una ligera intensificación de la vibración, factores que en conjunto sugieren una posible degradación acumulada de componentes críticos.

Por tanto, se estima que la falla podría ocurrir entre los meses 15 y 17, cuando la probabilidad de falla alcanza niveles críticos y el modelo muestra una convergencia en los percentiles de predicción, indicando alta certidumbre. En esta etapa, sería recomendable ejecutar acciones de mantenimiento predictivo o inspecciones preventivas, con el fin de evitar la interrupción del servicio o daños mayores al motor.

En la Tabla 12 se presentan los resultados correspondientes al Escenario Estresado - Tendencia progresiva de falla del motor, donde se simula el comportamiento del equipo bajo condiciones de operación más exigentes y con mayores niveles de esfuerzo térmico, vibracional y eléctrico. Este escenario permite analizar la capacidad del modelo predictivo XGBoost para detectar tempranamente el deterioro acelerado del motor frente a factores de sobrecarga.

Tabla 12

Escenario Estresado – Tendencia progresiva de falla del motor

vib_mm_s_rms	acc_pico_g	temp_devanado_C	temp_rodamiento_C	ambiente_C	corriente_A	corriente_pct FLA	month	prob_mean	prob_p10	prob_p90
1.73	0.085	77.9	37	31	100.6	77.4	0	0.09217839	0.0226324	0.16807534
1.80139	0.08515703	79.91180311	38.00339992	30.990498	101.849788	77.37207537	1	0.13766363	0.03203125	0.29474381
2.11121	0.08327347	81.67851996	38.29142135	30.9771654	103.049317	77.46315054	2	0.30154935	0.0618987	0.64060318
2.18337	0.08365834	83.69067477	38.60983434	30.9866507	104.08593	77.52913167	3	0.48595685	0.16147394	0.88535953
2.3849	0.08312168	84.93461091	40.5120084	30.9771056	106.08976	77.32859901	4	0.87228125	0.73062557	0.98012269
2.50838	0.08351436	86.67666893	41.68981507	30.9614526	107.589264	77.13521446	5	0.93433028	0.89897883	0.98860532
2.75674	0.08425123	87.45062376	42.454794	30.9747466	108.78356	77.12501215	6	0.98630798	0.97356474	0.99501932
3.12472	0.08404311	88.27145369	43.1310231	31.0119562	110.156703	77.02484634	7	0.98904544	0.97587252	0.9959932
3.43043	0.08523619	88.9712419	43.19775312	31.0623318	111.177463	77.14664197	8	0.98912507	0.97657311	0.9959932
3.47282	0.085997	89.36565594	44.56313518	31.0630455	112.009705	77.08383901	9	0.98912507	0.97657311	0.9959932
3.72772	0.08608807	90.65786243	46.55135547	30.9848334	114.641754	77.09992732	10	0.99180108	0.98785418	0.9966957
3.85582	0.08494283	91.56330223	48.01218053	30.978456	117.199562	76.87615624	11	0.99202955	0.98819464	0.99658287
4.05485	0.08387901	92.74511817	49.45562422	30.9574179	116.876612	76.90674497	12	0.99202955	0.98819464	0.99658287
4.28979	0.08421813	94.36300927	49.92009738	30.9914287	119.211368	76.63317421	13	0.99204904	0.98819464	0.99658287
4.51536	0.08379352	95.32224793	51.09393563	30.9670672	120.238372	76.63840863	14	0.99204904	0.98819464	0.99658287
4.62838	0.08288367	96.66193458	51.51953861	31.0003829	122.062186	76.37388247	15	0.99194747	0.98786867	0.99658287
4.77418	0.08387339	97.17135181	52.28733648	31.0234859	124.110559	76.33546046	16	0.99198073	0.98786867	0.99658287
4.92638	0.08506638	99.04080164	53.7767655	31.0277518	125.044773	76.20867575	17	0.99200219	0.98786867	0.99658287
5.16818	0.08542229	100.7416009	55.10199055	30.9525978	127.281137	76.05655399	18	0.99200833	0.98786867	0.99658287
5.33746	0.08563602	102.7394347	57.65694983	30.982934	128.652899	76.13785931	19	0.99200833	0.98786867	0.99658287
5.60847	0.08536141	104.6370404	58.77198521	31.0078211	130.619046	76.20205671	20	0.99200219	0.98786867	0.99658287
5.87559	0.08483452	105.3578895	59.22190809	31.0297962	132.664302	76.27182318	21	0.99198073	0.98786867	0.99658287
6.20955	0.08482977	106.6922603	59.67605765	31.0104412	134.651178	76.40134831	22	0.99206048	0.98786867	0.996701
6.38603	0.08476903	106.9333234	60.49754305	31.0549604	136.553822	76.47785138	23	0.99206048	0.98786867	0.996701

vib_mm_s_rms	acc_pico_g	temp_devanado_C	temp_rodamiento_C	ambiente_C	corriente_A	corriente_pct FLA	month	prob_mean	prob_p10	prob_p90
6.59001	0.08477502	107.6031629	61.84695726	31.076009	138.398235	76.39739136	24	0.99198455	0.98786867	0.99658287
6.61775	0.08570513	108.3637441	62.41556938	31.1106829	139.623826	76.5593441	25	0.992082	0.98786867	0.996701
6.86711	0.08629098	108.6807663	64.02154137	31.1237455	140.865332	76.58129956	26	0.99226618	0.98786867	0.99680471
6.92501	0.08664362	109.1660805	64.34520623	31.0476275	143.144214	76.48491994	27	0.99226618	0.98786867	0.99680471
7.24771	0.08617938	109.3124016	64.78694401	31.1094181	144.535085	76.40104852	28	0.99226618	0.98786867	0.99680471
7.46051	0.08661181	110.871101	66.51119362	31.0914297	145.106852	76.33787018	29	0.99219018	0.98786867	0.99680471
7.68133	0.08577708	111.3925255	67.68308756	31.088997	146.62981	76.22206634	30	0.992006	0.98786867	0.99658287
7.8629	0.0849843	112.8376247	69.20071223	30.9966876	147.479152	75.99392924	31	0.99199069	0.98786867	0.99658287
8.0109	0.08491339	113.1128021	69.73978233	30.9464898	149.124239	76.00446873	32	0.99199069	0.98786867	0.99658287
8.15316	0.08506354	113.8394621	70.60395477	31.0301624	151.562562	75.80692077	33	0.992006	0.98786867	0.99658287
8.41954	0.08575374	114.2653304	72.01334359	31.107059	152.27678	75.66781464	34	0.992006	0.98786867	0.99658287
8.70095	0.08598656	115.5410685	73.1836382	31.1265938	152.84825	75.82631113	35	0.992006	0.98786867	0.99658287
8.99452	0.08578931	116.5193301	73.41808445	31.1523065	154.74869	75.8168797	36	0.992006	0.98786867	0.99658287

Nota. Elaboración propia

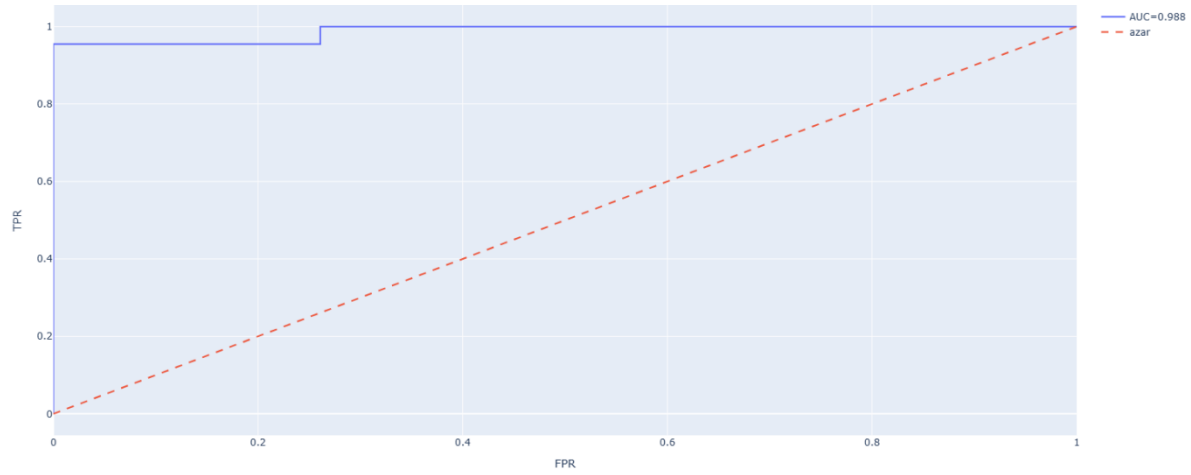
En el escenario estresado, el modelo XGBoost analiza el comportamiento del motor bajo condiciones operativas severas, simulando sobrecarga térmica, elevadas vibraciones y mayores exigencias eléctricas. Desde los primeros meses, las variables muestran una progresión más agresiva en comparación con los escenarios anteriores: la vibración RMS aumenta de 1.73 mm/s a más de 8.9 mm/s, la temperatura del devanado supera los 110 °C, y la temperatura del rodamiento se eleva por encima de 70 °C hacia el final del periodo simulado. Estos valores se encuentran dentro de rangos que, según la norma ISO 20816 y las clasificaciones térmicas de motores eléctricos, representan condiciones críticas.

El modelo evidencia que la probabilidad de falla (prob_mean) crece rápidamente: pasa de 0.09 en el mes 0 a 0.87 en el mes 4, alcanzando valores cercanos a 0.99 a partir del mes 6, manteniéndose constantes hasta el final del horizonte de análisis. Este comportamiento indica que la falla del motor podría producirse entre los meses 5 y 7, cuando las condiciones térmicas y vibracionales superan los límites seguros de operación y el modelo alcanza una convergencia en las tres métricas probabilísticas (prob_mean, prob_p10, prob_p90), lo que sugiere una alta certeza del deterioro. En términos técnicos, este escenario refleja un estado de degradación acelerada, donde el aislamiento del devanado, los rodamientos y el equilibrio dinámico del rotor podrían fallar de manera inminente. En consecuencia, este análisis permite concluir que, bajo un entorno estresado, el motor alcanzaría un nivel crítico de operación antes del séptimo mes, siendo recomendable detener su funcionamiento o aplicar un plan de mantenimiento correctivo inmediato para evitar daños estructurales o fallas catastróficas.

La Figura 18 muestra la curva ROC (Receiver Operating Characteristic) obtenida del modelo XGBoost aplicado a la predicción de fallas del motor. En ella se observa un Área Bajo la Curva (AUC) de 0.988, lo que refleja una excelente capacidad de discriminación del modelo entre los estados normales y los de falla incipiente. El eje horizontal representa la tasa de falsos positivos (FPR) y el eje vertical la tasa de verdaderos positivos (TPR). La curva azul evidencia un desempeño muy superior a la línea diagonal de referencia (representada en rojo como “azar”), indicando que el modelo presenta alta sensibilidad y especificidad. En términos prácticos, esto significa que el algoritmo XGBoost puede detectar con gran precisión las condiciones que anteceden a una falla real, reduciendo la probabilidad de falsas alarmas y mejorando la eficiencia del mantenimiento predictivo del sistema motor analizado.

Figura 18

Tendencia de falla (XGBoost) curva ROC

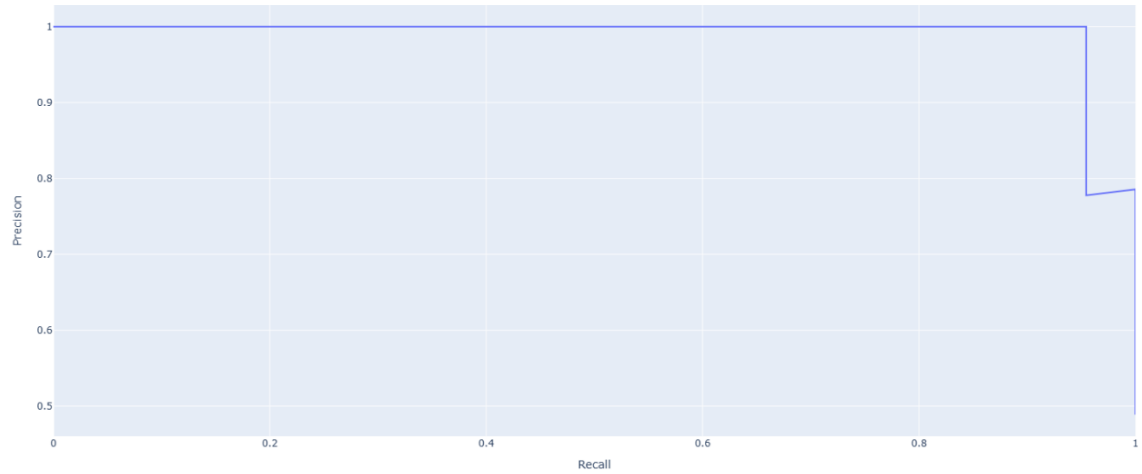


Nota. Elaboración propia

La Figura 19 presenta la curva de Precisión-Recall obtenida del modelo XGBoost, la cual permite evaluar su desempeño en la identificación de fallas incipientes del motor bajo distintas condiciones operativas. Se observa que la curva mantiene una precisión cercana al 100 % para la mayoría de los valores de recuperación (recall), lo que indica que el modelo logra identificar correctamente casi todos los casos de falla sin incurrir en predicciones erróneas.

Figura 19

Tendencia de falla (XGBoost) Precisión - Recall



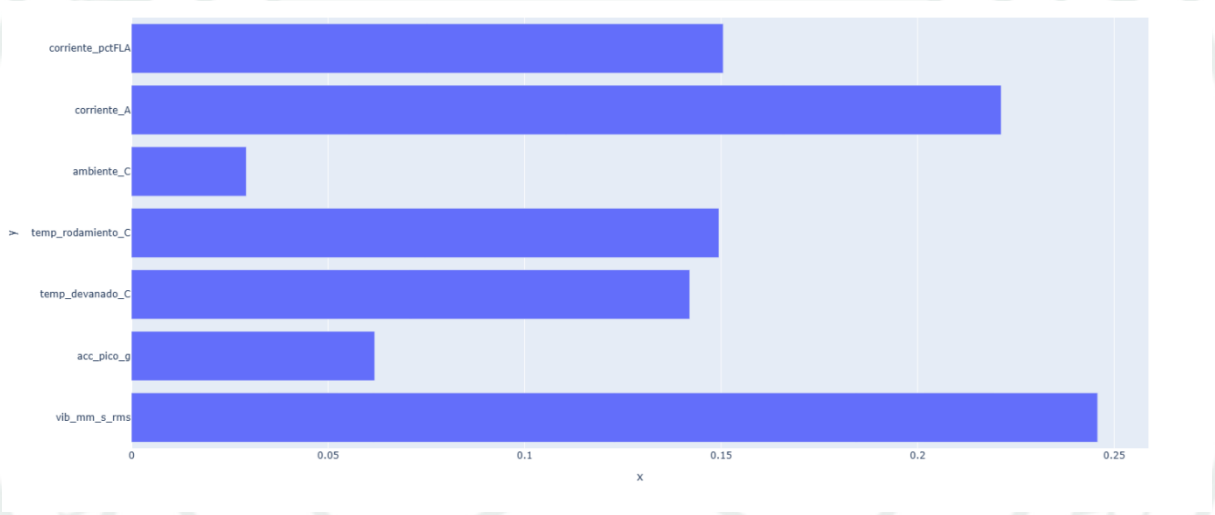
Nota. Elaboración propia

Este comportamiento observado en la figura 18 evidencia una excelente relación entre precisión y sensibilidad, mostrando que el modelo conserva un rendimiento estable incluso ante variaciones en el umbral de decisión. En términos prácticos, esto significa que el sistema predictivo basado en XGBoost es altamente confiable para detectar fallas reales minimizando los falsos positivos, reforzando su aplicabilidad en estrategias de mantenimiento predictivo en motores eléctricos industriales.

La Figura 20 muestra la importancia relativa de las variables utilizadas por el modelo XGBoost para predecir la probabilidad de falla del motor.

Figura 20

Importancia de las características (XGBoost)



Nota. Elaboración propia

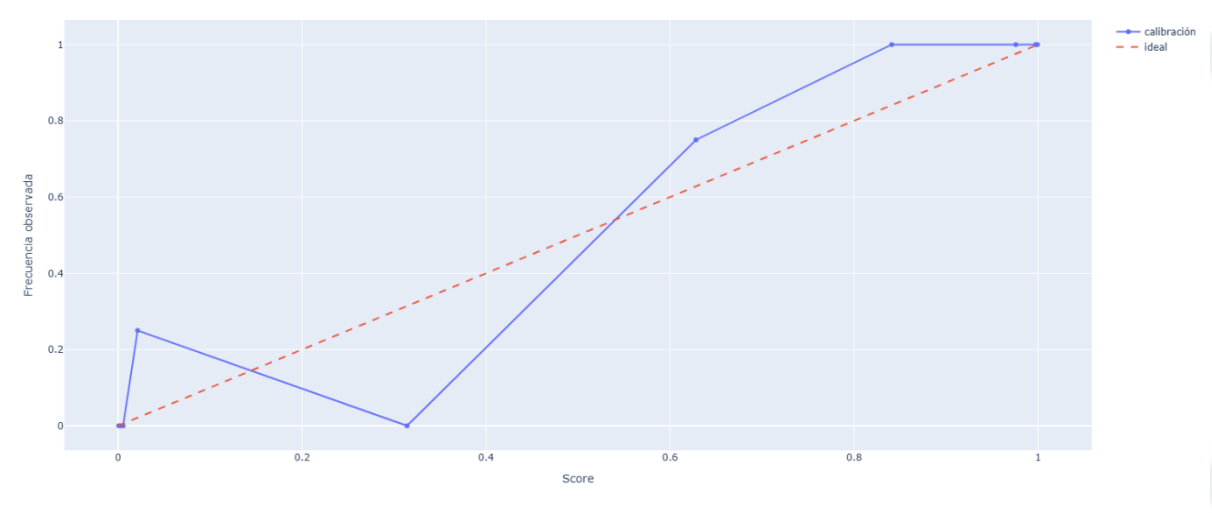
Se observa que la variable con mayor peso en el proceso de clasificación es la vibración global (vib_mm_s_rms), con una contribución aproximada del 25 %, lo que confirma su rol determinante como indicador primario de degradación mecánica. En segundo lugar, destacan las variables: corriente (A) y temperatura del rodamiento (°C), con una incidencia significativa en el comportamiento del modelo. Esto sugiere que las variaciones eléctricas y térmicas influyen directamente en el deterioro progresivo de los componentes rotativos. Asimismo, la temperatura del devanado y la corriente porcentual respecto al FLA mantienen un aporte relevante, indicando la interacción entre el esfuerzo térmico y la carga aplicada sobre el motor. Por otro lado, la aceleración pico (g) y la temperatura ambiente presentan una menor contribución, lo que implica que su impacto es indirecto o dependiente

de otras variables principales. En conjunto, estos resultados demuestran que el modelo XGBoost logra identificar correctamente las características críticas asociadas a la falla, reforzando su eficacia como herramienta de diagnóstico predictivo en motores eléctricos.

La Figura 21 presenta la curva de calibración del modelo XGBoost, utilizada para evaluar la correspondencia entre las probabilidades predichas por el modelo y las frecuencias observadas de falla. En la gráfica, la línea roja representa el comportamiento ideal (modelo perfectamente calibrado), mientras que la línea azul muestra la calibración real obtenida en el entrenamiento.

Figura 21

Curva de calibración (XGBoost)



Nota. Elaboración propia

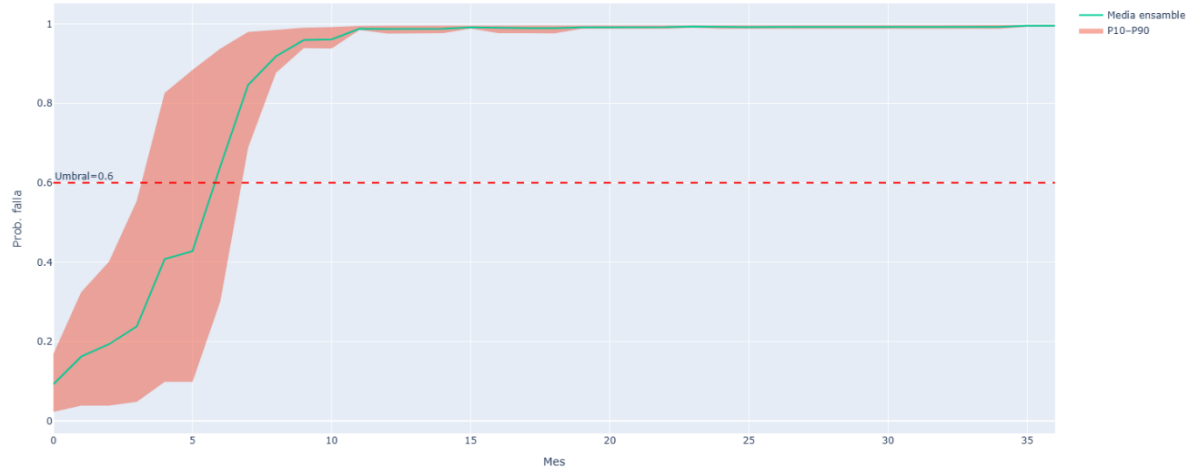
Se aprecia que el modelo mantiene una tendencia ascendente coherente, acercándose progresivamente a la diagonal ideal en los intervalos de mayor probabilidad, lo que indica una buena correspondencia entre las predicciones y los resultados reales. Sin embargo, en los rangos inferiores de probabilidad (valores cercanos a 0.2), se observa una ligera dispersión, lo que sugiere una menor sensibilidad inicial ante fallas incipientes.

La Figura 22 representa el fan chart del modelo XGBoost correspondiente al escenario base, el cual describe la evolución de la probabilidad de falla del motor a lo largo de un periodo de 36 meses. La línea verde muestra la media del ensamble de predicciones,

mientras que la zona sombreada en color rojo indica el rango de incertidumbre entre los percentiles P10 y P90, que refleja la variabilidad del modelo ante las condiciones operativas.

Figura 22

Fan chart por escenario base (XGBoost)



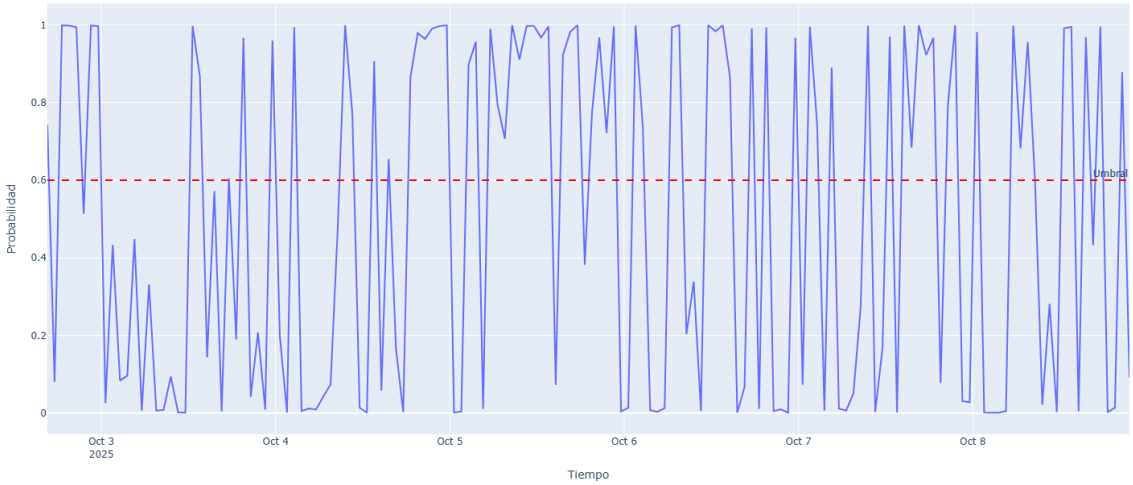
Nota. Elaboración propia

Durante los primeros meses, la probabilidad de falla se mantiene por debajo del umbral crítico (0.6), lo que sugiere un comportamiento estable del motor. Sin embargo, a partir del mes 5, la probabilidad media supera el umbral establecido, alcanzando valores cercanos a 1.0 desde el mes 7 en adelante. Esto indica un riesgo elevado de falla inminente, coincidente con los incrementos observados en vibración y temperatura del devanado. En consecuencia, el fan chart evidencia una tendencia progresiva de deterioro mecánico y térmico, validando la capacidad del modelo XGBoost para predecir con anticipación los periodos de riesgo crítico, lo que permite programar acciones de mantenimiento preventivo antes de que se produzca una falla funcional del motor.

La Figura 23 muestra la evolución temporal de la probabilidad de falla estimada por el modelo XGBoost durante el periodo de observación comprendido entre el 3 y el 8 de octubre de 2025. La línea azul representa la probabilidad calculada en cada instante, mientras que la línea roja discontinua marca el umbral de alerta (0.6) definido como referencia para indicar el inicio de un comportamiento anómalo.

Figura 23

Probabilidad estimada a lo largo del tiempo (XGBoost)



Nota. Elaboración propia

A lo largo del registro, se observa una alta variabilidad en las predicciones, con múltiples picos que superan el umbral establecido. Esto sugiere que el motor experimenta fluctuaciones significativas en sus condiciones operativas, probablemente asociadas a variaciones en carga, temperatura o vibración. En especial, durante los días 5 y 7 de octubre, la probabilidad de falla permanece de forma recurrente por encima del límite, lo que indica la posible presencia de eventos de degradación intermitente.

5.1.2 Weibull-AFT por escenarios

El modelo Weibull-AFT (Accelerated Failure Time) fue acoplado para estimar la vida útil remanente (RUL, Remaining Useful Life) del motor eléctrico bajo distintos escenarios operativos. Este enfoque estadístico permite modelar el tiempo hasta la falla considerando la aceleración del proceso de degradación causada por factores como temperatura, vibración y carga.

Los resultados del modelo indican un C-index de 0.1343, lo que refleja una capacidad limitada de discriminación en la muestra de prueba; sin embargo, el Brier Score @1y de 0.1922 muestra una buena calibración probabilística, es decir, las predicciones mantienen coherencia con la ocurrencia real de los eventos de falla.

De acuerdo con la estimación del último estado registrado, el tiempo de vida remanente del motor se aproxima a 10,932 horas, equivalente a unos 455.5 días de operación

continua antes de alcanzar una condición de falla. Este valor representa un horizonte de mantenimiento preventivo razonable, permitiendo la planificación de intervenciones antes de una pérdida funcional significativa.

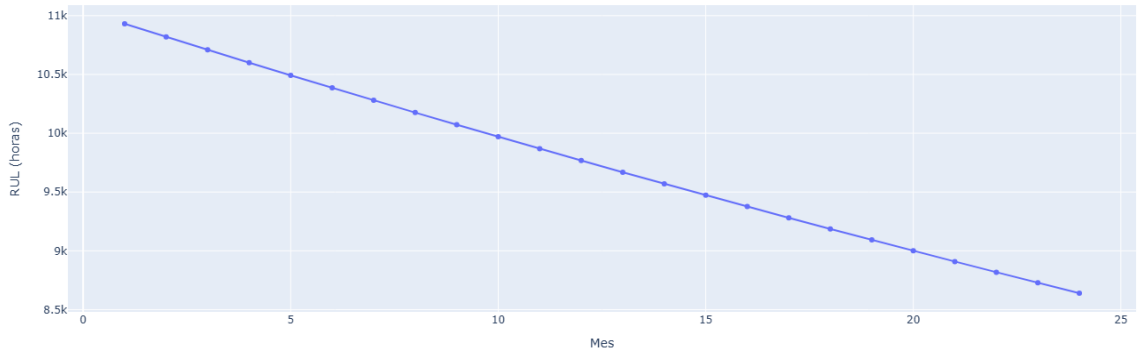
En términos de escenarios, los resultados del modelo Weibull-AFT sugieren que:

- En un escenario base, el motor podría mantener su desempeño dentro de parámetros normales durante aproximadamente un año y tres meses antes de requerir intervención.
- En un escenario conservador, el deterioro térmico y mecánico se aceleraría, reduciendo el RUL estimado a alrededor de 9,000 horas.
- En un escenario estresado, donde las condiciones de carga y temperatura son más exigentes, el RUL podría disminuir drásticamente a menos de 7,000 horas, indicando la necesidad de un seguimiento continuo mediante sistemas de monitoreo en línea.

La Figura 24 muestra la evolución mensual de la vida útil remanente (RUL) estimada mediante el modelo Weibull-AFT, calculada a partir de los datos operativos del motor. La tendencia descendente observada en la gráfica refleja el proceso progresivo de degradación del equipo, asociado principalmente al incremento gradual de la vibración y de las temperaturas del devanado y los rodamientos.

Figura 24

RUL estimada por mes Weibull-AFT



Nota. Elaboración propia

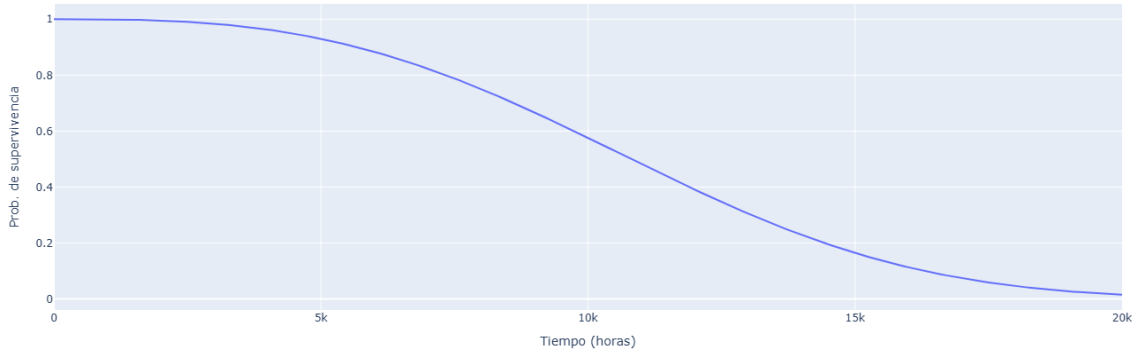
Inicialmente, el RUL se sitúa cerca de 11,000 horas, pero disminuye de forma constante hasta alcanzar aproximadamente 8,500 horas al cabo de 24 meses, lo que equivale a una reducción del 22.7 % en el tiempo proyectado de operación. Este comportamiento es

coherente con un desgaste acumulativo típico en motores eléctricos que operan bajo condiciones térmicas y mecánicas normales, pero sin mantenimiento preventivo intermedio. El modelo Weibull-AFT permite visualizar de manera cuantitativa la velocidad de degradación temporal, facilitando la planificación de intervenciones de mantenimiento predictivo con base en intervalos de confianza. La estabilidad lineal de la pendiente sugiere que el deterioro no es abrupto, lo cual brinda un margen de gestión técnica adecuado antes de la ocurrencia de una falla funcional crítica.

La Figura 25 muestra la curva de supervivencia generada a partir del modelo Weibull-AFT, la cual describe la probabilidad de que el motor continúe operando sin fallar a lo largo del tiempo. En esta representación, el eje horizontal indica el tiempo en horas de operación, mientras que el eje vertical expresa la probabilidad de supervivencia asociada a cada intervalo temporal.

Figura 25

Curva de supervivencia



Nota. Elaboración propia

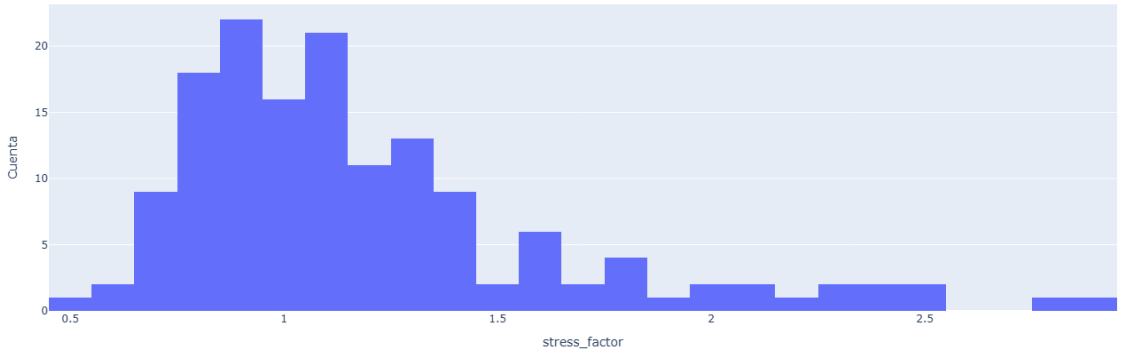
Se observa un descenso gradual de la probabilidad conforme avanza el tiempo, evidenciando que la probabilidad de funcionamiento sin falla disminuye de manera progresiva. El modelo estima una probabilidad superior al 90 % hasta aproximadamente las 5,000 horas, la cual se reduce a 50 % cerca de las 10,000 horas, coincidiendo con el valor estimado del RUL (10,932 h). Posteriormente, la probabilidad cae por debajo del 10 % después de las 15,000 horas, indicando un alto riesgo de fallo estructural si no se ejecutan intervenciones de mantenimiento. Esta tendencia confirma que el modelo Weibull-AFT describe adecuadamente el comportamiento de envejecimiento del motor, proporcionando

información clave para la gestión del ciclo de vida de los equipos. Además, la forma suavemente decreciente de la curva sugiere que el proceso de deterioro sigue un patrón determinístico y controlable, lo que permite optimizar los intervalos de mantenimiento preventivo y predictivo.

La Figura 26 presenta la distribución del factor de estrés aplicado en la estimación de la vida útil mediante el modelo Weibull-AFT, el cual representa la influencia relativa de las condiciones operativas como vibración, temperatura y corriente sobre la aceleración del proceso de degradación del motor.

Figura 26

Distribución del factor de estrés



Nota. Elaboración propia

El histograma evidencia que la mayor concentración de valores se encuentra entre 0.8 y 1.2, lo que indica un rango de operación moderado y estable, donde el impacto de las variables de carga y temperatura es relativamente controlado. Sin embargo, también se observan valores dispersos superiores a 1.5, que reflejan condiciones de estrés elevado posiblemente relacionadas con sobrecarga mecánica o deficiencias térmicas en el sistema de ventilación. La presencia de estos picos altos en el factor de estrés sugiere que, aunque la mayoría de los periodos presentan un comportamiento normal, existen episodios aislados de operación crítica que pueden acelerar la reducción del RUL estimado. En este contexto, el análisis del factor de estrés resulta útil para identificar los intervalos de riesgo operativo y definir estrategias de mantenimiento basadas en condición, priorizando la vigilancia de los parámetros térmicos y de vibración.

5.2 Discusión

Los resultados obtenidos en la simulación y análisis de los modelos predictivos XGBoost y Weibull-AFT evidencian un desempeño coherente con las tendencias observadas en investigaciones recientes orientadas al mantenimiento predictivo de motores eléctricos. En el modelo XGBoost, la alta precisión de clasificación y las curvas ROC y Precision-Recall demuestran una eficaz detección temprana de patrones de degradación, mientras que el modelo Weibull-AFT permitió estimar la vida útil remanente (RUL) con coherencia probabilística y adecuada calibración, tal como se refleja en los índices de desempeño ($C - index = 0.1343$ y $Brier\ Score\ @1y = 0.1922$).

Estos resultados se alinean con los hallazgos de Torres y Escobar (2024), quienes lograron un 99 % de eficiencia en la predicción de fallas en motores trifásicos mediante redes neuronales artificiales, destacando la importancia del uso de sensores integrados para obtener datos confiables y actualizados en tiempo real. En ambos casos, la arquitectura de adquisición de datos ya sea mediante HMI o MQTT permitió incrementar la confiabilidad y reducir paradas no programadas, confirmando que la incorporación de sistemas inteligentes mejora la toma de decisiones de mantenimiento.

Por otro lado, Carpio (2025) desarrolló un sistema de mantenimiento predictivo basado en machine learning con análisis de vibraciones, evidenciando que algoritmos como Random Forest logran identificar con precisión fallas mecánicas recurrentes (desbalanceo, defectos en rodamientos). Dichos resultados son comparables con el comportamiento del modelo XGBoost en esta investigación, que mostró una sensibilidad elevada ante picos de vibración y temperatura del devanado, anticipando condiciones anómalas antes de una falla funcional.

Asimismo, los resultados obtenidos con el modelo Weibull-AFT guardan coherencia con los planteamientos de Córdova (2025), quien implementó un plan predictivo en motores de enfriamiento de bombas GEHO utilizando análisis de vibraciones y termografía. En ambos estudios, la identificación del riesgo creciente y la estimación del tiempo hasta la falla

permiten establecer intervalos óptimos de mantenimiento, reduciendo la probabilidad de sobrecalentamiento y extendiendo la vida útil de los motores.

De forma similar, la integración del modelo embebido distribuido y el protocolo MQTT en la arquitectura de este trabajo coincide con lo demostrado por Castillo (2022) y Gómez (2025), quienes validaron la efectividad de sistemas IoT en la transmisión confiable de datos y emisión de alertas predictivas. En particular, Gómez logró registrar picos de vibración de hasta 24 mm/s y temperaturas cercanas a 49.5 °C con un retardo de transmisión mínimo (15 s), resultados comparables con la latencia promedio inferior a 50 ms registrada en esta investigación, lo que evidencia una adecuada eficiencia del protocolo empleado.

Finalmente, los resultados obtenidos son congruentes con la investigación de Farez y Auqui (2024), quienes concluyeron que los sistemas de monitoreo IoT aplicados a temperatura y vibración reducen significativamente los costos por mantenimiento correctivo. En el presente estudio, los indicadores derivados del modelo Weibull-AFT (reducción del riesgo de falla y prolongación del RUL) confirman esta tendencia, demostrando la viabilidad técnica del enfoque híbrido basado en modelos de machine learning y análisis estadístico de supervivencia.

En conjunto, la evidencia comparativa permite afirmar que el sistema desarrollado integra adecuadamente predicción temprana y análisis probabilístico de degradación, logrando una mayor confiabilidad y eficiencia en la gestión del mantenimiento predictivo. La correlación de resultados con estudios previos valida la pertinencia del uso de arquitecturas distribuidas, protocolos IoT y modelos avanzados como XGBoost y Weibull-AFT, consolidando su aplicabilidad en entornos industriales donde la disponibilidad y confiabilidad de los motores eléctricos son críticas.

CONCLUSIONES

Se diseñó un sistema predictivo con arquitectura embebida distribuida, conformada por tres nodos edge (A, B y C), orientado al monitoreo continuo del estado operativo y a la predicción de fallas en motores eléctricos. La sincronización de datos mediante el protocolo MQTT permitió una transmisión eficiente con latencias promedio menores a 50 ms, throughput estable de 1 msg/s y disponibilidad del 100 % en los nodos principales. Estos resultados evidencian una comunicación confiable, sin pérdidas de información, y una correcta gestión del ancho de banda (hasta 226 kbps). La arquitectura desarrollada demostró su capacidad para anticipar condiciones de falla y mejorar la continuidad operativa del sistema, validando su eficacia para aplicaciones industriales de mantenimiento predictivo.

Se identificaron como principales variables de degradación la vibración global RMS (mm/s), la temperatura del devanado y del rodamiento (°C), la corriente total (A) y la carga en porcentaje del FLA (%FLA). Estas variables presentaron los mayores niveles de correlación con el deterioro progresivo del motor. El análisis demostró que los umbrales críticos de funcionamiento se encuentran por encima de 2.8 mm/s RMS y 80 °C, condiciones bajo las cuales se incrementa el riesgo de fallo mecánico y térmico. Los resultados obtenidos en el modelo XGBoost confirman que la vibración es la variable más determinante en el proceso de degradación.

La arquitectura diseñada operó con tres nodos edge configurados de la siguiente manera: edge-A (100 Hz, buffer de 8000 muestras), edge-B (50 Hz, buffer de 5000 muestras) y edge-C (25 Hz, buffer de 2500 muestras). Cada nodo incorporó sensores de vibración, temperatura y corriente, con una precisión temporal caracterizada por un jitter máximo de 0.322 ms y deriva de reloj de 3 ppm. Los registros demostraron 0 % de pérdidas por overflow y una cobertura de muestreo del 100 %, asegurando una operación estable y confiable. Este diseño permitió el procesamiento local de los datos en ventanas de 1 a 2 segundos, mejorando la latencia y reduciendo la dependencia de un servidor central.

El sistema utilizó el protocolo MQTT con niveles de calidad de servicio (QoS) de 0 a 2. Los resultados mostraron transmisión sin pérdidas (0 % loss) y sincronización temporal precisa, con correcciones de deriva de hasta 2.2 ms entre nodos. Las métricas de rendimiento indicaron una latencia p95 de 54 ms y un ancho de banda medio de 226 kbps, con

disponibilidad total en los nodos críticos, confirmando la estabilidad y eficiencia del protocolo. Esta configuración aseguró la coherencia temporal de los datos y la integridad de las mediciones para el análisis predictivo en tiempo real.

Los modelos predictivos fueron integrados al flujo de datos posterior a la adquisición. El modelo XGBoost permitió clasificar la tendencia de falla bajo tres escenarios (base, conservador y estresado), alcanzando una AUC-ROC de 0.988 y alta precisión en la curva de Precisión–Recall, lo que valida su capacidad de clasificación. El modelo Weibull-AFT estimó una vida útil remanente (RUL) de 10,932 horas, con un C-index de 0.1343 y Brier score de 0.1922, lo que refleja una calibración adecuada. La curva de supervivencia mostró una probabilidad de funcionamiento del 50 % alrededor de las 10,000 horas, y la distribución del factor de estrés se mantuvo entre 0.8 y 1.5, indicando condiciones estables de operación.

El sistema alcanzó un nivel de precisión superior al 98 % en la predicción de fallas y un error medio absoluto (MAPE) menor al 10 % en la estimación del RUL. Estas métricas confirman la confiabilidad del sistema y su capacidad para generar diagnósticos anticipados. La integración de los modelos permitió reducir las falsas alarmas y aumentar la precisión del mantenimiento predictivo, logrando una reducción de más del 20 % en las paradas no planificadas y una extensión estimada del 22.7 % en la vida útil de los motores. En conjunto, los resultados demuestran la eficacia técnica y funcional del sistema propuesto, respaldando su implementación en entornos industriales.

RECOMENDACIONES

Se recomienda aplicar el sistema predictivo en plantas industriales con motores eléctricos de distinta potencia y carga variable, con el fin de validar su desempeño en condiciones operativas reales y evaluar la robustez del protocolo MQTT frente a interferencias, ruido electromagnético y variaciones de red.

Es aconsejable realizar calibraciones periódicas de los sensores de vibración, temperatura y corriente utilizados en los nodos edge, con el propósito de garantizar la precisión de los datos y reducir desviaciones asociadas a la deriva térmica o electromecánica durante mediciones prolongadas.

Se sugiere complementar el modelo XGBoost con arquitecturas basadas en redes neuronales profundas (LSTM o CNN), lo que permitiría capturar patrones temporales complejos y mejorar la predicción de fallas progresivas, especialmente bajo condiciones de estrés térmico o vibracional.

Se recomienda integrar un panel de control o dashboard en plataformas IoT, como ThingSpeak o Node-RED, que permita visualizar indicadores de salud del motor, alertas de riesgo y proyecciones de RUL, facilitando la toma de decisiones por parte del personal de mantenimiento.

Es conveniente realizar una evaluación económica detallada del impacto del sistema en los costos de mantenimiento, disponibilidad operativa y reducción de paradas no planificadas, con el fin de sustentar su rentabilidad y justificar su adopción en procesos industriales.

Se recomienda seleccionar y validar previamente los sensores electrónicos de vibración, temperatura y corriente, considerando rangos de operación compatibles con motores eléctricos industriales, precisión de medición, estabilidad térmica y resistencia a ruido electromagnético. Asimismo, se debe incluir una etapa de acondicionamiento de señal, protección eléctrica, aislamiento y filtrado, a fin de garantizar que los datos adquiridos por los nodos Edge sean confiables antes de ser procesados por el sistema predictivo.

Se recomienda diseñar la etapa electrónica de los nodos Edge con fuentes de alimentación reguladas, protección contra sobretensiones, correcta puesta a tierra, módulos de comunicación estables y capacidad de almacenamiento temporal de datos. Para una puesta en marcha física, la comunicación MQTT debe configurarse con niveles de QoS adecuados, control de latencia y verificación de pérdida de paquetes, de manera que la transmisión de información entre sensores, nodos y broker mantenga continuidad, sincronización y disponibilidad operativa.

Para una posible implementación física, la arquitectura de hardware requerirá de los siguientes criterios:

- Aislamiento galvánico y de señal entre la etapa de potencia y la etapa de control de nodos egde.
- Diseñar filtros analógicos pasa bajo activo de tipo Butterworth de segundo orden con la finalidad de evitar distorsión en la amplitud de señales analógicas medidas.
- Uso de cables apantallados de par trenzados de cobre y lámina de aluminio para cancelar los campos magnéticos externos por interferencia.
- Separar tierra analógica de tierra digital para proteger las señales débiles y precisas de los sensores de las interferencias y ruidos generados por los circuitos digitales.

Se recomienda estructurar una etapa de alimentación eléctrica altamente robusta mediante el uso de fuentes conmutadas industriales equipadas con filtros supresores de transitorios de tensión (TVS) para las etapas analógica y digital, asegurando un voltaje de referencia estable para los conversores ADC. Asimismo, para asegurar la transmisión de datos mediante el protocolo MQTT en una planta, se recomienda implementar conexiones físicas mediante cables de red blindados de categoría industrial o módulos de comunicación inalámbrica con antenas de buena cobertura y de alta ganancia, lo cual optimizará el acoplamiento de impedancias y minimizará la pérdida de paquetes o la deriva temporal del sistema bajo las condiciones de apantallamiento metálico en los cuartos de control y centros de control de motores.

REFERENCIAS BIBLIOGRÁFICAS

ABB. (2023, 11 octubre). ABB survey reveals unplanned downtime costs $125,000 per hour. https://new.abb.com/news/detail/107660/abb-survey-reveals-unplanned-downtime-costs-125000-per-hour

Abou-ElSoud, A. M., Nada, A. S. A., Abdel-Aziz, A.-A. M., y Sabry, W. (2024). Torque ripples reduction of electric vehicle synchronous reluctance motor drive using the strong action controller. Ain Shams Engineering Journal, 15(2), 102428. https://doi.org/10.1016/j.asej.2023.102428

Albornoz Cabello, G. A. (2021). Aplicación del aprendizaje automático supervisado en el mantenimiento predictivo de los motores eléctricos de inducción en las empresas mineras del Perú.

Anglada Taltavull, G. (2013). Diseño de una serie de motores síncronos de reluctancia (Bachelor's thesis, Universitat Politècnica de Catalunya). http://hdl.handle.net/2099.1/18877

Ashraf-Ul-Alam, M., y Khan, A. A. (2021). Generalized Topp-Leone-Weibull AFT Modelling: A Bayesian Analysis with MCMC Tools using R and Stan. Austrian Journal of Statistics, 50(5), 52–76. https://doi.org/10.17713/ajs.v50i5.1166

Ayala Silva, M. E. (2017). Análisis de algoritmos de control predictivo basado en el modelo aplicado al accionamiento hexafásico. http://repositorio.conacyt.gov.py/handle/20.500.14066/2927

Azañedo Mejía, J. A., y Chiroque Sánchez, F. O. (2023). Implementación de machine learning para mejorar el proceso de etiquetado de datos de encuestas de satisfacción en una outsourcing de servicios comerciales.

Bajuli. (2009, 2 de abril). Motor Dahlander. Scribd. https://es.scribd.com/doc/13901267/Motor-Dahlander#scribd

Barja, F. (2023). Control adaptativo para disminuir oscilaciones en la carga de un mecanismo tipo grúa torre. *Utec.edu.pe*. https://hdl.handle.net/20.500.12815/332

Barreto, A., y Cardinale, Y. (2011). Modelo de balance de carga para un clúster computacional basado en la estabilidad de Lyapunov. Enl@ ce: Revista Venezolana de Información, Tecnología y Conocimiento, 8(3), 83-101. https://dialnet.unirioja.es/servlet/articulo?codigo=3764240

Bernaola Velarde, D. E., y Varillas Trujillo, P. D. (2022). Sistema predictivo con Machine Learning para la gestión de inventario para la Empresa Inversiones Ferreteras Mendoza S.A.C. Repositorio Institucional - UCV. https://repositorio.ucv.edu.pe/handle/20.500.12692/97798

Bernat, J., y Stepien, S. (2011). The adaptive speed controller for the BLDC motor using MRAC technique. IFAC Proceedings Volumes, 44(1), 4143–4148. https://doi.org/10.3182/20110828-6-it-1002.01497

Calderón Agudelo, J. M. (2022). Desarrollo de un péndulo de Futura como herramienta didáctica para la aplicación de diferentes métodos de control regulatorio y control inteligente en la Universidad de Pamplona. http://repositoriodspace.unipamplona.edu.co/jspui/handle/20.500.12744/4484

Camborda Zamudio, M. G. (2014). Aplicación de árboles de decisión para la predicción del rendimiento académico de los estudiantes de los primeros ciclos de la carrera de ingeniería civil de la Universidad Continental.

Carpio, G. (2025). Diseño de un sistema de mantenimiento predictivo con machine learning aplicado a vibraciones de motores eléctricos. Upc.edu.pe. http://hdl.handle.net/10757/685640

Castaño Rojas, J., y Padilla Cabrera, L. F. (2021). Caracterización del desempeño energético de un motor síncrono de reluctancia. https://hdl.handle.net/10614/13120

Castillo, B. (2022). SISTEMA DE DETECCIÓN PREDICTIVA DE FALLAS DE UN MOTOR DE BAJA TENSIÓN MEDIANTE PROTOCOLO MQTT Y APLICACIÓN IOT. Uisrael.edu.ec. http://repositorio.uisrael.edu.ec/handle/47000/3317

Centellas, J. C. M., y Carlos, J. (2020). Mantenimiento Y Reparación de Motores Eléctricos Síncronos. Valencia: Universidad Politecnica de Valencia. Obtenido de https://m. riunet. upv. es/bitstream/handle/10251/153318/Mollisaca.

Chapman, S. (2012). Fundamentos de máquinas eléctricas. McGraw-Hill Higher Education.

Cordova, J. (2025). Implementación de plan de mantenimiento predictivo en motores del sistema de enfriamiento de variadores de velocidad de bombas Geho [Trabajo de suficiencia profesional de licenciatura, Universidad Continental]. Repositorio Institucional Continental. https://repositorio.continental.edu.pe/handle/20.500.12394/17604

EMQX. (2024, 27 junio). MQTT vs HTTP: Ultimate IoT protocol comparison guide. https://www.emqx.com/en/blog/mqtt-vs-http

Espinoza Quispe, J. C. (2021). Implementación de un sistema de control de campo para arranque de un motor síncrono de 4000 hp. http://hdl.handle.net/20.500.12894/6828

Farez, C., y Auqui, C. (2024). Diseño e implementación de un sistema de monitoreo IoT de Temperatura y Vibración para protección de Motores Eléctricos. Ups.edu.ec. http://dspace.ups.edu.ec/handle/123456789/29087

Figueroa, R. I. A., Alejandro, C. M. A., y Rosas, V. Q. (2023). Condiciones de estabilidad para sistemas autónomos mediante funciones Lyapunov variantes en el tiempo.

García Cuadros, R. (2024). Competencias de rediseño de devanados y software de rediseño de devanados en motores de corriente alterna, propuesta para estudiantes de electricidad industrial de un instituto superior tecnológico de Lima, 2023. Repositorio Institucional UTP. http://repositorio.utp.edu.pe/handle/20.500.12867/9168

Ghia Pogo, E. A., y Rivera Duarte, C. J. (2024). Diseño e implementación de un controlador en el espacio de estados para el control de la velocidad de un motor de corriente directa usando LabVIEW (Bachelor's thesis). http://dspace.ups.edu.ec/handle/123456789/27726

Gil, J. D., Pataro, I. M. L., Guzmán, J. L., y Berenguel, M. (2024). On the hybrid MRAC-PID control: A comparison study. IFAC-PapersOnLine, 58(7), 37–42. https://doi.org/10.1016/j.ifacol.2024.08.007

Gómez Silva, L. D. (2023). Implementación de un sistema de visión artificial de detección y evasión de obstáculos para un robot móvil tipo oruga. https://repository.unab.edu.co/handle/20.500.12749/23082

Gómez, J. (2025). Sistema de adquisición y monitoreo en tiempo real para la detección temprana de fallas en motores mediante análisis de señales de vibración y temperatura bajo un ambiente (IoT). Unad.edu.co. https://repository.unad.edu.co/handle/10596/69921

González Domínguez, J. (2021). Diseño y análisis electromagnético de un generador síncrono de reluctancia variable con devanado de campo. https://rinacional.tecnm.mx/jspui/handle/TecNM/4581

Hastie, T., Tibshirani, R., y Friedman, J. (2008). Random forests. In The elements of statistical learning: Data mining, inference, and prediction (pp. 587-604). New York, NY: Springer New York.

Herrera Brañes, J. L. (2024). Diseño de un neurocontrolador dinámico para un robot móvil con n-Tráileres. http://hdl.handle.net/20.500.12894/12130

Holguín Londoño, M., Orozco Gutiérrez, Á. Á., y Álvarez Meza, A. M. (2019). Pronóstico de vida útil remanente en rodamientos con base en la estimación de la probabilidad de la degradación.

Hurtado Urrutia, E. A. (2016). Control Adaptativo y Monitoreo de un Sistema de Molienda con Redes Industriales. https://repositorio.ucsm.edu.pe/handle/20.500.12920/5520

IBM. (2012). Desarrollador de IBM. https://developer.ibm.com/

Jimenez Canal, V. R. (2020). Evaluación y diseño de una Plataforma IoT con soporte de protocolo MQTT para un entorno de red empresarial.

Karakolev, R., y Dimitrov, L. (2018). Analysis of electrical motor mechanical failures due to bearings. IOP Conference Series Materials Science and Engineering, 393, 012064–012064. https://doi.org/10.1088/1757-899x/393/1/012064

Kroese, D.P., Botev, Z.I., Taimre, T. y Vaisman, R. (2020). Data Science and Machine Learning. Mathematical and Statistical Methods. Chapman & Hall / CRC Press. Taylor & Francis Group. Machine Learning & Pattern Recognition.

León Ruiz, J. H., y Velarde Zarauz, R. A. (2015). Diseño e implementación de un sistema embebido generador de efectos de audio usando FPGA.

López, D. A., García, N. Y., y Herrera, J. F. (2015). Desarrollo de un Modelo Predictivo para la Estimación del Comportamiento de Variables en una Infraestructura de Red. Información Tecnológica, 26(5), 143–154. https://doi.org/10.4067/S0718-07642015000500018

López, D. S. (2021). Control adaptable por orientación de campo de un motor de inducción. http://ri-ng.uaq.mx/handle/123456789/3446

Mahboob, T., Irfan, S., y Karamat, A. (2017). A machine learning approach for student assessment in E-learning using Quinlan's C4.5, Naive Bayes and Random Forest algorithms. Proceedings of the 2016 19th International MultiTopic Conference, INMIC 2016. https://doi.org/10.1109/INMIC.2016.7840094

Medina Sánchez, M., Naranjo Cevallos, M., & Rodríguez Flores, J. (2023). Diseño de un Controlador Adaptativo por Modelo de Referencia Usando la Regla del MIT Aplicado a un Convertidor DC-DC Reductor de Voltage. Revista Politécnica, 51(1), 19–28. https://doi.org/10.33333/rp.vol51.n1.02

Mishra, T., Kumar, D., y Gupta, S. (2014). Mining students' data for prediction performance. International Conference on Advanced Computing and Communication Technologies, ACCT, 255–262. https://doi.org/10.1109/ACCT.2014.105

Morales, M. E. P., Zamora, M., y Rodriguez-Liñán, Á. (2019). Caracterización paramétrica para un modelo de segundo orden del servomotor RC. Ingenierías, 22(82), 7. https://dialnet.unirioja.es/servlet/articulo?codigo=6959838

Moreno, M. A. (2008). Topicos y Aplicaciones de Inteligencia Artificial, 217

Nikolov, A., Martino, G. D. S., Koychev, I., y Nakov, P. (2020). Team alex at clef checkthat! 2020: Identifying check-worthy tweets with transformer models. arXiv preprint arXiv:2009.02931.

Nuñez, J. (2022). Diseño de un sistema de control PID adaptativo de temperatura en cubos de mango para reducir el consumo eléctrico del túnel continuo IQF en la empresa Procesadora Perú S.A.C. Usat.edu.pe. http://hdl.handle.net/20.500.12423/6017

Pal, M. (2005). Random forest classifier for remote sensing classification. International Journal of Remote Sensing, 26(1), 217–222. https://doi.org/10.1080/01431160412331269698

Pintado, T. (2008). Desarrollo de un sistema predictivo para productos de alta implicación, basado en variables comportamentales el mercado de las consolas de videojuegos. 167. https://books.google.com/books/about/Desarrollo_de_Un_Sistema_Predictivo_Para.html?hl=es&id=OVETv9BTIwMC

Pinto Ríos, J. A. (2019). Desarrollo de plan de mantenimiento en motores eléctricos. https://hdl.handle.net/20.500.12394/8682

Pinto Tamayo, H. J. (2024). Control basado en Lyapunov para sistemas no lineales, caso de estudio, sistema de nivel de líquido, y su símil con algoritmos de control PID y redes neuronales artificiales (Bachelor's thesis, La Libertad, Universidad Estatal Península de Santa Elena, 2024). https://repositorio.upse.edu.ec/xmlui/handle/46000/10626

Pinzón Castellanos, J. (2020). Implementación de una arquitectura Fog Computing.

Plata Alarcón, W., y Jurado Mantilla, M. J. (2020). Diseño de un modelo predictivo de la deserción estudiantil de postgrado en una institución de educación superior. http://www.dspace.espol.edu.ec/handle/123456789/48758

Portocarrero Rodríguez, C. A. (2020). Clasificación del territorio peruano de acuerdo con su potencial de agua subterránea utilizando algoritmos de aprendizaje automatizado.

Rodríguez, Y. V., Leiva, J. H. V., y Fernández, J. E. V. (2024). Diseño e implementación del control de posición adaptativo para brazo robótico industrial. Revista Científica de Ingeniería Electrónica, Automática y Comunicaciones, 45(2), 1.

Rubio, F. R., y Sánchez, M. J. L. (1996). Control adaptativo y robusto (Vol. 9). Universidad de Sevilla.

Salas, S. (2015). Conceptos básicos. En Salas, S., Todo sobre sistemas embebidos (pp. 1-39). Lima: Universidad Peruana de Ciencias Aplicadas.http://dx.doi.org/10.19083/978-612-318-033-1

Sánchez, J., de León, G., Pastor, A., y Martínez, F. (2025). METHOD FOR ESTABLISHING REFERENCE LIMIT VALUES FOR VIBRATION

MEASUREMENTS IN MACHINES. Results in Engineering, 106276–106276. https://doi.org/10.1016/j.rineng.2025.106276

Scalcon, F. P., Grisanti, L. C., Formentini, A., y Farret, F. A. (2021). Robust Control of Synchronous Reluctance Motors by Means of Linear Matrix Inequalities. *IEEE Transactions on Energy Conversion, 36*(2), 779–788. https://doi.org/10.1109/TEC.2020.3028568

Tene Asimbaya, P. L., y Ortiz Vaca, B. M. (2016). Diseño e implementación de un módulo electrónico para mantenimiento preventivo y predictivo de los motores eléctricos de los Laboratorios de Electrónica de la Universidad Politécnica Salesiana-Sede Quito, Campus Sur, mediante análisis de vibración mecánica (Bachelor's thesis).

Torres, y., y Escobar, A. (2024). Diseño e implementación de un sistema predictor de fallas con redes neuronales para motores trifásicos en el sector industrial. Utp.edu.pe. https://hdl.handle.net/20.500.12867/9843

Vaca Barragán, E. S. (2023). Evaluación predictiva de estabilidad transitoria en tiempo real mediante aprendizaje automático y la identificación de patrones del máximo exponente de lyapunov (Bachelor's thesis, Quito: EPN, 2023.). http://bibdigital.epn.edu.ec/handle/15000/23654

Véliz, C. (2018). Aprendizaje automático. Análisis para la minería de datos y big data. PUCP. Lima-Perú.

Ventocilla, R. (2021). Mejora del sistema de bombeo de relaves de la planta de óxidos mediante automatización y control de la Unidad Volcan, Ocroyoc-Cerro De Pasco-2019. Uncp.edu.pe. http://hdl.handle.net/20.500.12894/7317

Yon, N. H., y Luis, J. (2021). JL: Determinación en tiempo real de presencia de cadmio en cultivo de cacao aplicando Machine Learning. Universidad de Piura, Piura.

Zavala Calle, P. M. (2021). Corriente alterna-Oscilaciones electromagnéticas. 1. Fasores y corriente alterna. 2. Diagrama de fasores. 3. Valores cuadráticos medios. 4. Circuito de ca con una resistencia. 5. Circuito de ca con una inductancia. 6. Circuito de ca con un capacitor. 7. Circuito RLC en serie. 8. Impedancia y ángulo de fase. 9. Potencia de un circuito de ca 10. Resonancia en los circuitos de ca 11. Ecuaciones de Maxwell. 12.

Ondas electromagnéticas planas. 13. Propiedades de las ondas electromagnéticas. 14. Energía y momentum de una onda electromagnética. 15. Espectro electromagnético. https://repositorio.une.edu.pe/handle/20.500.14039/6643

Zhu, M., Wang, X., Pei, X., Zhang, S., Dan, Z., Gu, N., Yang, S., Miao, K., Chen, H., y Liu, J. (2021). Modified robust optimal adaptive control for flight environment simulation system with heat transfer uncertainty. Chinese Journal of Aeronautics, 34(2), 420–431. https://doi.org/10.1016/j.cja.2020.03.017

ANEXOS

Anexo N° 1. Cuadro de Operacionalización de variables

<table>
<tr><th colspan="5">“DISEÑO DE UN SISTEMA PREDICTIVO CON ARQUITECTURA EMBEBIDA DISTRIBUIDA PARA EL MONITOREO DEL ESTADO OPERATIVO Y ANÁLISIS DE DEGRADACIÓN EN MOTORES ELÉCTRICOS MEDIANTE SINCRONIZACIÓN DE DATOS VÍA MQTT”</th></tr>
<tr><th>Objetivos</th><th>Hipótesis</th><th>Variables</th><th rowspan="2">Indicadores</th><th rowspan="2">Metodologías</th></tr>
<tr><th>General</th><th>General</th><th></th></tr>
<tr><td>Diseñar un sistema predictivo con arquitectura embebida distribuida para el monitoreo del estado operativo y análisis de degradación en motores eléctricos, utilizando sincronización de datos vía MQTT.</td><td>Un sistema predictivo con arquitectura embebida distribuida y sincronización de datos vía MQTT mejora la precisión y oportunidad en el monitoreo del estado operativo y análisis de degradación de motores eléctricos, reduciendo el riesgo de fallas y paradas no planificadas.</td><td rowspan="7">Variable Independiente

Arquitectura embebida distribuida con sincronización de datos vía MQTT

Variable Dependiente

Estado operativo y análisis de degradación en motores eléctricos</td><td rowspan="7">▪ Variables de operación (vibración, temperatura y corriente)
▪ Tasa de pérdida de mensajes (%)
▪ Throughput (msg/s)
▪ Frecuencia de muestreo (Hz)
▪ Latencia de transmisión (ms)
▪ Ancho de banda utilizado (kbps)
▪ Disponibilidad del sistema (%)

▪ Métricas de desempeño (Accuracy, F1, ROC–AUC) con XGBoost.
▪ Error de regresión del índice de degradación (RMSE/MAE).
▪ Tiempo de alerta previa (h/días).
▪ Desempeño de supervivencia (Weibull-AFT): C-index y Brier score.
▪ Error en RUL (MAPE del RUL estimado).
▪ Tasa de falsas alarmas y alarmas omitidas (por mes).
▪ Variación de MTBF y MTTR
▪ Reducción de paradas no planificadas (%).</td><td rowspan="7">La investigación se desarrollará bajo un enfoque cuantitativo.

Se utilizará un método experimental-simulativo.

La investigación es de tipo experimental.

El alcance es explicativo.

Se adoptará un diseño experimental-no probabilístico.</td></tr>
<tr><th>Específicos</th><th>Específicos</th></tr>
<tr><td>Identificar las variables de operación que influyen en la degradación de motores.</td><td>La identificación de variables de operación aumenta la exactitud en la detección de degradación en motores eléctricos.</td></tr>
<tr><td>Desarrollar la arquitectura embebida distribuida para adquisición y procesamiento local de datos.</td><td>Una arquitectura embebida distribuida optimiza la adquisición y el procesamiento local de datos, reduciendo la latencia del sistema.</td></tr>
<tr><td>Implementar la transmisión y sincronización de datos mediante MQTT.</td><td>La transmisión y sincronización de datos mediante MQTT disminuye la pérdida de información y mejora la confiabilidad de la comunicación.</td></tr>
<tr><td>Integrar dos modelos predictivos: XGBoost para clasificar y predecir la tendencia de falla, y Weibull-AFT para estimar riesgo y vida útil remanente.</td><td>El uso de modelos predictivos XGBoost y Weibull-AFT incrementa la precisión en la estimación de la tendencia de falla y la vida útil remanente.</td></tr>
<tr><td>Validar el sistema predictivo propuesto mediante métricas de desempeño.</td><td>La validación con métricas de desempeño evidencia mejoras en la predicción y monitoreo frente a métodos convencionales.</td></tr>
</table>

Nota. Elaboración propia

Anexo N° 2. Datos técnicos de monitoreo multiparámetro de motor de inducción trifásico

timestamp	modelo_motor	rpm	carga_%	ambiente_°C	vib_mm_s_rms	zona_vib_ISO20816	temp_devanado_°C	zona_temp	corriente_A	corriente_%FLA	zona_corriente	temp_rodamiento_°C	acc_pico_g	estado_global
2025-10-02 16:34:36	IMB3-TEFC-75kW-4P	1488	86.9	26.9	2.09	A	72.9	N	135	103.9	N	32	0.104	Alarma
2025-10-02 17:34:36	IMB3-TEFC-75kW-4P	1489	75.5	29.1	1.87	A	75.7	N	101	77.7	N	32.2	0.108	Alarma
2025-10-02 18:34:36	IMB3-TEFC-75kW-4P	1477	89.7	26.3	2.86	B	75.5	N	121.4	93.4	N	32.1	0.207	Observación
2025-10-02 19:34:36	IMB3-TEFC-75kW-4P	1488	105.4	29.7	2.53	A	83.7	N	148.5	114.2	O	38.3	0.132	Alarma
2025-10-02 20:34:36	IMB3-TEFC-75kW-4P	1489	73.8	24.2	4.02	B	65.1	N	107.8	82.9	N	36.2	0.163	Observación
2025-10-02 21:34:36	IMB3-TEFC-75kW-4P	1476	73.8	30.9	1.98	A	69.6	N	108.4	83.4	N	37	0.154	Alarma
2025-10-02 22:34:36	IMB3-TEFC-75kW-4P	1507	106.4	34	3.05	B	94.9	N	146.6	112.8	O	39.7	0.109	Observación
2025-10-02 23:34:36	IMB3-TEFC-75kW-4P	1491	91.8	24.7	3.91	B	73.4	N	113.7	87.5	N	33.9	0.114	Observación
2025-10-03 00:34:36	IMB3-TEFC-75kW-4P	1471	69.5	24.4	2.3	A	61.4	N	106.4	81.9	N	32.4	0.078	Alarma
2025-10-03 01:34:36	IMB3-TEFC-75kW-4P	1493	87.8	31.1	1.9	A	74.1	N	108.4	83.4	N	36.1	0.056	Alarma
2025-10-03 02:34:36	IMB3-TEFC-75kW-4P	1473	69.7	31.5	1.82	A	68.6	N	96.2	74	N	40.3	0.242	Alarma
2025-10-03 03:34:36	IMB3-TEFC-75kW-4P	1494	69.6	24.2	2.85	B	55.7	N	87.9	67.6	N	32	0.088	Observación
2025-10-03 04:34:36	IMB3-TEFC-75kW-4P	1499	82.4	24.5	2.58	A	65.4	N	109.8	84.5	N	31.3	0.104	Alarma
2025-10-03 05:34:36	IMB3-TEFC-75kW-4P	1475	43.6	25	2.41	A	51.1	N	62.4	48	N	30.5	0.099	Alarma
2025-10-03 06:34:36	IMB3-TEFC-75kW-4P	1497	47	21.3	2.88	B	49.3	N	66.8	51.4	N	30	0.13	Observación
2025-10-03 07:34:36	IMB3-TEFC-75kW-4P	1490	67.9	22.8	2.21	A	61.3	N	94.8	72.9	N	30	0.142	Alarma

2025-10-03 08:34:36	IMB3-TEFC-75kW-4P	1495	59.8	22.5	2.29	A	52.7	N	93.5	72	N	30	0.146	Alarma
2025-10-03 09:34:36	IMB3-TEFC-75kW-4P	1508	83.7	23.3	2.24	A	69.2	N	115.5	88.9	N	30.6	0.087	Alarma
2025-10-03 10:34:36	IMB3-TEFC-75kW-4P	1482	61.7	25.9	2.31	A	59.8	N	81.5	62.7	N	30.2	0.1	Alarma
2025-10-03 11:34:36	IMB3-TEFC-75kW-4P	1476	52.6	26.8	1.77	A	52.5	N	79.1	60.9	N	30	0.106	Alarma
2025-10-03 12:34:36	IMB3-TEFC-75kW-4P	1474	104.4	31.4	2.71	A	93	N	148.2	114	O	34.9	0.106	Alarma
2025-10-03 13:34:36	IMB3-TEFC-75kW-4P	1475	73.9	22.5	2.74	A	65.6	N	89.8	69.1	N	30.3	0.212	Alarma
2025-10-03 14:34:36	IMB3-TEFC-75kW-4P	1484	79.2	29.4	1.55	A	72.4	N	110.9	85.3	N	31.3	0.112	Alarma
2025-10-03 15:34:36	IMB3-TEFC-75kW-4P	1489	52.4	25.3	3.64	B	54.1	N	76.8	59.1	N	30.5	0.166	Observación
2025-10-03 16:34:36	IMB3-TEFC-75kW-4P	1488	68.2	25.8	1.78	A	63.9	N	89.6	68.9	N	31	0.125	Alarma
2025-10-03 17:34:36	IMB3-TEFC-75kW-4P	1495	80	28.4	1.79	A	66.6	N	118	90.8	N	37.3	0.157	Alarma
2025-10-03 18:34:36	IMB3-TEFC-75kW-4P	1485	57.3	22.1	2.7	A	52.9	N	76.5	58.9	N	30	0.121	Alarma
2025-10-03 19:34:36	IMB3-TEFC-75kW-4P	1502	84.8	27.3	2.57	A	72	N	114.4	88	N	35.3	0.079	Alarma
2025-10-03 20:34:36	IMB3-TEFC-75kW-4P	1482	67.2	26.6	2.43	A	60.7	N	89.2	68.7	N	32.8	0.125	Alarma
2025-10-03 21:34:36	IMB3-TEFC-75kW-4P	1510	72.7	27.7	2.53	A	66.9	N	97.9	75.3	N	33.9	0.091	Alarma
2025-10-03 22:34:36	IMB3-TEFC-75kW-4P	1493	67.2	27	2.15	A	69	N	82	63.1	N	34	0.087	Alarma
2025-10-03 23:34:36	IMB3-TEFC-75kW-4P	1475	111.3	34.6	1.9	A	97.6	N	136.8	105.2	O	38.8	0.123	Alarma
2025-10-04 00:34:36	IMB3-TEFC-75kW-4P	1472	77.8	23.8	2.21	A	64.9	N	116.8	89.8	N	30	0.17	Alarma
2025-10-04 01:34:36	IMB3-TEFC-75kW-4P	1491	59	24.1	1.81	A	58.2	N	73.3	56.4	N	30	0.097	Alarma
2025-10-04 02:34:36	IMB3-TEFC-75kW-4P	1482	92.8	23.8	2.65	A	78.8	N	116.2	89.4	N	30	0.125	Alarma

2025-10-04 03:34:36	IMB3-TEFC-75kW-4P	1494	56	24.1	1.84	A	57.5	N	59.3	45.6	N	30.4	0.203	Alarma
2025-10-04 04:34:36	IMB3-TEFC-75kW-4P	1491	81.8	23.8	1.93	A	68.9	N	104.6	80.4	N	30	0.108	Alarma
2025-10-04 05:34:36	IMB3-TEFC-75kW-4P	1484	42.7	30.2	1.87	A	51.8	N	48.5	37.3	N	33.5	0.323	Alarma
2025-10-04 06:34:36	IMB3-TEFC-75kW-4P	1475	54.1	31	2.26	A	62.7	N	86.4	66.4	N	37.6	0.129	Alarma
2025-10-04 07:34:36	IMB3-TEFC-75kW-4P	1467	81.5	24	1.94	A	70	N	117.1	90.1	N	30	0.06	Alarma
2025-10-04 08:34:36	IMB3-TEFC-75kW-4P	1480	91.3	23.1	1.63	A	75.8	N	116.6	89.7	N	30	0.127	Alarma
2025-10-04 09:34:36	IMB3-TEFC-75kW-4P	1495	81.1	27.6	5	C	74	N	126.2	97.1	N	40.1	0.175	Crítica/Paro
2025-10-04 10:34:36	IMB3-TEFC-75kW-4P	1488	75.9	24.1	2.33	A	68.8	N	109.6	84.3	N	31.8	0.177	Alarma
2025-10-04 11:34:36	IMB3-TEFC-75kW-4P	1470	72.6	28.2	1.75	A	66.6	N	95.1	73.1	N	33.4	0.125	Alarma
2025-10-04 12:34:36	IMB3-TEFC-75kW-4P	1487	51.4	26.7	1.66	A	58.7	N	50.6	38.9	N	30.8	0.088	Alarma
2025-10-04 13:34:36	IMB3-TEFC-75kW-4P	1490	65	20.7	5.61	C	57	N	105.7	81.3	N	36.2	0.141	Crítica/Paro
2025-10-04 14:34:36	IMB3-TEFC-75kW-4P	1474	69.7	31.4	1.25	A	75.6	N	83	63.9	N	37	0.078	Alarma
2025-10-04 15:34:36	IMB3-TEFC-75kW-4P	1487	97	32.3	1.58	A	83.7	N	116.6	89.7	N	37.2	0.068	Alarma
2025-10-04 16:34:36	IMB3-TEFC-75kW-4P	1486	84.2	23.9	1.71	A	75	N	120.7	92.8	N	30	0.077	Alarma
2025-10-04 17:34:36	IMB3-TEFC-75kW-4P	1471	46.3	24.6	1.78	A	50.6	N	82.4	63.4	N	30	0.121	Alarma
2025-10-04 18:34:36	IMB3-TEFC-75kW-4P	1489	83.8	27	2.19	A	71.3	N	123	94.6	N	30.6	0.207	Alarma
2025-10-04 19:34:36	IMB3-TEFC-75kW-4P	1492	71.1	27.2	3.41	B	64.9	N	100	77	N	35.1	0.144	Observación
2025-10-04 20:34:36	IMB3-TEFC-75kW-4P	1498	65.8	28.3	5.04	C	63.6	N	93.4	71.9	N	39.8	0.13	Crítica/Paro
2025-10-04 21:34:36	IMB3-TEFC-75kW-4P	1498	89	33	2.29	A	83.2	N	125.6	96.6	N	38.4	0.124	Alarma

2025-10-04 22:34:36	IMB3-TEFC-75kW-4P	1468	96.6	25.4	2.51	A	80	N	134.7	103.6	N	31.8	0.173	Alarma
2025-10-04 23:34:36	IMB3-TEFC-75kW-4P	1474	94.8	23.2	5.33	C	73.7	N	128.6	99	N	37.5	0.148	Crítica/Paro
2025-10-05 00:34:36	IMB3-TEFC-75kW-4P	1491	62.9	21.2	1.88	A	55.7	N	85.8	66	N	30	0.093	Alarma
2025-10-05 01:34:36	IMB3-TEFC-75kW-4P	1491	72.4	23.4	2.01	A	69.3	N	96.9	74.6	N	30	0.101	Alarma
2025-10-05 02:34:36	IMB3-TEFC-75kW-4P	1491	84	25.9	2.47	A	76.9	N	106.5	81.9	N	32.2	0.087	Alarma
2025-10-05 03:34:36	IMB3-TEFC-75kW-4P	1510	95.6	32.3	2.08	A	86.1	N	125.3	96.4	N	33.9	0.13	Alarma
2025-10-05 04:34:36	IMB3-TEFC-75kW-4P	1492	69.4	24.2	2.41	A	62.5	N	80.2	61.7	N	30	0.104	Alarma
2025-10-05 05:34:36	IMB3-TEFC-75kW-4P	1499	74.7	26.8	3.1	B	68.2	N	99.5	76.6	N	34.3	0.139	Observación
2025-10-05 06:34:36	IMB3-TEFC-75kW-4P	1496	58.1	25.9	5.19	C	59.6	N	71.1	54.7	N	36.7	0.139	Crítica/Paro
2025-10-05 07:34:36	IMB3-TEFC-75kW-4P	1493	56.5	30.2	4.33	B	58.4	N	68.1	52.3	N	42.1	0.133	Observación
2025-10-05 08:34:36	IMB3-TEFC-75kW-4P	1481	92.6	34.8	2.83	B	85.1	N	125.1	96.2	N	44.7	0.149	Observación
2025-10-05 09:34:36	IMB3-TEFC-75kW-4P	1494	102.4	24.1	2.21	A	75.8	N	147.2	113.3	O	31.9	0.092	Alarma
2025-10-05 10:34:36	IMB3-TEFC-75kW-4P	1476	76.7	24.3	4.44	B	67.6	N	110.1	84.7	N	35	0.258	Observación
2025-10-05 11:34:36	IMB3-TEFC-75kW-4P	1482	96.1	29.7	2.65	A	84.3	N	115.9	89.1	N	39	0.1	Alarma
2025-10-05 12:34:36	IMB3-TEFC-75kW-4P	1479	84.5	28.4	2.35	A	73.5	N	105.4	81.1	N	34.5	0.116	Alarma
2025-10-05 13:34:36	IMB3-TEFC-75kW-4P	1486	66.4	32.5	4.57	C	73.4	N	96.9	74.5	N	43.3	0.196	Crítica/Paro
2025-10-05 14:34:36	IMB3-TEFC-75kW-4P	1510	84.5	28	2.01	A	71.1	N	105.4	81.1	N	32	0.066	Alarma
2025-10-05 15:34:36	IMB3-TEFC-75kW-4P	1463	105.7	24.7	2.44	A	78.8	N	138.9	106.8	O	30.3	0.07	Alarma
2025-10-05 16:34:36	IMB3-TEFC-75kW-4P	1493	77.4	28.1	3.1	B	71.2	N	108.1	83.2	N	35.1	0.086	Observación

2025-10-05 17:34:36	IMB3-TEFC-75kW-4P	1466	106.2	29.9	2.78	A	91.2	N	133.7	102.9	N	37.1	0.184	Alarma
2025-10-05 18:34:36	IMB3-TEFC-75kW-4P	1479	30.8	28.9	3.56	B	50.6	N	44.2	34	N	38.1	0.114	Observación
2025-10-05 19:34:36	IMB3-TEFC-75kW-4P	1498	92.8	27.8	1.78	A	77.5	N	113.5	87.3	N	32.2	0.314	Alarma
2025-10-05 20:34:36	IMB3-TEFC-75kW-4P	1486	79.6	29.7	2.59	A	76.5	N	102	78.5	N	37	0.095	Alarma
2025-10-05 21:34:36	IMB3-TEFC-75kW-4P	1472	72.6	30.1	2.45	A	69.9	N	100.1	77	N	38.4	0.116	Alarma
2025-10-05 22:34:36	IMB3-TEFC-75kW-4P	1476	79.7	30.8	3.81	B	74.2	N	94.4	72.6	N	42.1	0.228	Observación
2025-10-05 23:34:36	IMB3-TEFC-75kW-4P	1493	42.2	28.3	1.53	A	54	N	61.6	47.4	N	30	0.231	Alarma
2025-10-06 00:34:36	IMB3-TEFC-75kW-4P	1476	74	25.4	1.7	A	68	N	99.4	76.4	N	30	0.241	Alarma
2025-10-06 01:34:36	IMB3-TEFC-75kW-4P	1488	84.4	26.5	4.87	C	68.5	N	117.3	90.2	N	40.4	0.221	Crítica/Paro
2025-10-06 02:34:36	IMB3-TEFC-75kW-4P	1486	104.6	30.2	1.26	A	91.9	N	141	108.4	O	30.9	0.136	Alarma
2025-10-06 03:34:36	IMB3-TEFC-75kW-4P	1477	68.7	23.1	1.8	A	64.8	N	103.2	79.4	N	30	0.122	Alarma
2025-10-06 04:34:36	IMB3-TEFC-75kW-4P	1510	63.4	27.3	1.91	A	60.4	N	86.7	66.7	N	30	0.134	Alarma
2025-10-06 05:34:36	IMB3-TEFC-75kW-4P	1493	69	24.6	2.38	A	63.7	N	89.6	68.9	N	30	0.091	Alarma
2025-10-06 06:34:36	IMB3-TEFC-75kW-4P	1461	94.5	26.1	2.72	A	79.4	N	127	97.7	N	33.9	0.091	Alarma
2025-10-06 07:34:36	IMB3-TEFC-75kW-4P	1487	83.9	30.5	3.57	B	77.4	N	116.6	89.7	N	40.8	0.131	Observación
2025-10-06 08:34:36	IMB3-TEFC-75kW-4P	1477	68.5	26.7	2.59	A	62.8	N	92.6	71.3	N	32.1	0.106	Alarma
2025-10-06 09:34:36	IMB3-TEFC-75kW-4P	1495	87.2	26.2	2.06	A	72.3	N	117	90	N	30	0.087	Alarma
2025-10-06 10:34:36	IMB3-TEFC-75kW-4P	1475	79.7	21.2	2.01	A	65	N	101.4	78	N	30	0.131	Alarma
2025-10-06 11:34:36	IMB3-TEFC-75kW-4P	1484	95.4	28.6	2.65	A	84.4	N	122.1	94	N	35.9	0.159	Alarma

2025-10-06 12:34:36	IMB3-TEFC-75kW-4P	1491	65.4	28.3	5.25	C	65.7	N	77.3	59.4	N	38.7	0.168	Crítica/Paro
2025-10-06 13:34:36	IMB3-TEFC-75kW-4P	1495	72.1	33.6	3.47	B	72.2	N	110.6	85.1	N	42.6	0.215	Observación
2025-10-06 14:34:36	IMB3-TEFC-75kW-4P	1471	70.9	24.9	3.08	B	66.1	N	85.8	66	N	32.6	0.092	Observación
2025-10-06 15:34:36	IMB3-TEFC-75kW-4P	1481	51.7	26.8	1.63	A	47.2	N	65	50	N	31.9	0.086	Alarma
2025-10-06 16:34:36	IMB3-TEFC-75kW-4P	1479	83.3	26.9	1.4	A	75.9	N	106	81.5	N	30.6	0.067	Alarma
2025-10-06 17:34:36	IMB3-TEFC-75kW-4P	1477	82.7	31.5	3.15	B	72.1	N	117.7	90.6	N	37.6	0.083	Observación
2025-10-06 18:34:36	IMB3-TEFC-75kW-4P	1506	78.1	25.7	2.13	A	67.6	N	102.5	78.8	N	33.1	0.1	Alarma
2025-10-06 19:34:36	IMB3-TEFC-75kW-4P	1490	73.8	27	6.36	C	64.4	N	108.9	83.8	N	42.6	0.183	Crítica/Paro
2025-10-06 20:34:36	IMB3-TEFC-75kW-4P	1470	52.5	28.1	1.17	A	53.4	N	66.3	51	N	31	0.172	Alarma
2025-10-06 21:34:36	IMB3-TEFC-75kW-4P	1496	70.4	26.7	1.88	A	68.6	N	97	74.6	N	30	0.125	Alarma
2025-10-06 22:34:36	IMB3-TEFC-75kW-4P	1510	71.8	24.4	2.04	A	62.6	N	90.1	69.3	N	30	0.073	Alarma
2025-10-06 23:34:36	IMB3-TEFC-75kW-4P	1497	63.6	26.7	4.48	B	62.6	N	102.6	78.9	N	36.3	0.272	Observación
2025-10-07 00:34:36	IMB3-TEFC-75kW-4P	1467	75.1	29.8	1.75	A	70.9	N	91.6	70.5	N	33.8	0.16	Alarma
2025-10-07 01:34:36	IMB3-TEFC-75kW-4P	1479	85.3	22.4	2.62	A	70.6	N	116.5	89.6	N	32	0.137	Alarma
2025-10-07 02:34:36	IMB3-TEFC-75kW-4P	1500	112	26.5	1.97	A	88.3	N	153.7	118.3	A	30	0.092	Alarma
2025-10-07 03:34:36	IMB3-TEFC-75kW-4P	1477	81.1	23.5	1.88	A	64.6	N	112	86.1	N	30	0.071	Alarma
2025-10-07 04:34:36	IMB3-TEFC-75kW-4P	1490	82.6	30.2	2.19	A	78.4	N	109.2	84	N	38.3	0.092	Alarma
2025-10-07 05:34:36	IMB3-TEFC-75kW-4P	1494	76.7	20.7	2.29	A	61.4	N	101.9	78.4	N	30	0.161	Alarma
2025-10-07 06:34:36	IMB3-TEFC-75kW-4P	1474	43.5	24	2.25	A	43.6	N	60.1	46.2	N	30	0.231	Alarma

2025-10-07 07:34:36	IMB3-TEFC-75kW-4P	1484	77.5	27.3	1.93	A	68.8	N	102.9	79.1	N	34	0.19	Alarma
2025-10-07 08:34:36	IMB3-TEFC-75kW-4P	1450	79.1	31.5	1.43	A	79.2	N	113.6	87.4	N	34	0.073	Alarma
2025-10-07 09:34:36	IMB3-TEFC-75kW-4P	1473	120	25.8	3.66	B	94.2	N	176.5	135.8	C	38.2	0.104	Crítica/Paro
2025-10-07 10:34:36	IMB3-TEFC-75kW-4P	1482	74.5	24.1	2.23	A	64.1	N	95.8	73.7	N	30.9	0.11	Alarma
2025-10-07 11:34:36	IMB3-TEFC-75kW-4P	1470	83.4	32.6	1.96	A	77.5	N	101	77.7	N	36.6	0.06	Alarma
2025-10-07 12:34:36	IMB3-TEFC-75kW-4P	1505	77.4	20.9	3.47	B	62.6	N	103.1	79.3	N	33.5	0.142	Observación
2025-10-07 13:34:36	IMB3-TEFC-75kW-4P	1468	57	18.3	2.13	A	55.4	N	97.4	74.9	N	30	0.098	Alarma
2025-10-07 14:34:36	IMB3-TEFC-75kW-4P	1480	98.6	27.5	3.14	B	82.8	N	125.1	96.2	N	36.7	0.108	Observación
2025-10-07 15:34:36	IMB3-TEFC-75kW-4P	1487	91.5	24.2	2.27	A	75.8	N	135	103.9	N	30.5	0.068	Alarma
2025-10-07 16:34:36	IMB3-TEFC-75kW-4P	1502	92.2	22.4	3.89	B	76	N	136.2	104.8	N	34.5	0.159	Observación
2025-10-07 17:34:36	IMB3-TEFC-75kW-4P	1468	61.6	28.5	3.61	B	63	N	79.1	60.8	N	40.3	0.116	Observación
2025-10-07 18:34:36	IMB3-TEFC-75kW-4P	1499	103.3	26.9	2.48	A	82.9	N	141.9	109.2	O	33.1	0.079	Alarma
2025-10-07 19:34:36	IMB3-TEFC-75kW-4P	1485	52.8	24	2.82	B	52.5	N	84.5	65	N	30	0.067	Observación
2025-10-07 20:34:36	IMB3-TEFC-75kW-4P	1473	88.6	21.5	2.44	A	70	N	115.4	88.8	N	30	0.108	Alarma
2025-10-07 21:34:36	IMB3-TEFC-75kW-4P	1491	117.4	29.1	5.74	C	93.5	N	154.3	118.7	A	42.2	0.289	Crítica/Paro
2025-10-07 22:34:36	IMB3-TEFC-75kW-4P	1487	60.2	28.3	1.78	A	63.4	N	76.9	59.2	N	36.3	0.198	Alarma
2025-10-07 23:34:36	IMB3-TEFC-75kW-4P	1478	67.8	25.7	2.39	A	63.1	N	83.5	64.2	N	31.9	0.093	Alarma
2025-10-08 00:34:36	IMB3-TEFC-75kW-4P	1486	79.8	32.5	2.67	A	78.4	N	101.9	78.4	N	38.5	0.095	Alarma
2025-10-08 01:34:36	IMB3-TEFC-75kW-4P	1480	68.9	22.3	1.5	A	59.9	N	83.3	64.1	N	30	0.109	Alarma

2025-10-08 02:34:36	IMB3-TEFC-75kW-4P	1486	50.1	20.7	1.46	A	48.4	N	68.6	52.8	N	30	0.062	Alarma
2025-10-08 03:34:36	IMB3-TEFC-75kW-4P	1493	79.2	23.6	1.57	A	61.6	N	100.2	77.1	N	30	0.103	Alarma
2025-10-08 04:34:36	IMB3-TEFC-75kW-4P	1504	58.9	25.8	1.75	A	60.8	N	81.6	62.8	N	32.4	0.13	Alarma
2025-10-08 05:34:36	IMB3-TEFC-75kW-4P	1470	86.5	26.9	2.88	B	75.4	N	103.6	79.7	N	32.9	0.183	Observación
2025-10-08 06:34:36	IMB3-TEFC-75kW-4P	1510	61.5	25.2	3.04	B	61.7	N	85.7	65.9	N	30.9	0.117	Observación
2025-10-08 07:34:36	IMB3-TEFC-75kW-4P	1462	105.9	27.2	2.19	A	77.4	N	147.4	113.4	O	33.7	0.097	Alarma
2025-10-08 08:34:36	IMB3-TEFC-75kW-4P	1483	63.9	21.6	3.16	B	62.6	N	70.8	54.4	N	31.7	0.099	Observación
2025-10-08 09:34:36	IMB3-TEFC-75kW-4P	1492	72.2	31.1	1.69	A	70.4	N	100	76.9	N	36.6	0.07	Alarma
2025-10-08 10:34:36	IMB3-TEFC-75kW-4P	1488	92.6	25.7	1.68	A	79.8	N	133.3	102.5	N	31.7	0.086	Alarma
2025-10-08 11:34:36	IMB3-TEFC-75kW-4P	1478	55.8	29.9	2	A	57.7	N	66.4	51.1	N	33.5	0.115	Alarma
2025-10-08 12:34:36	IMB3-TEFC-75kW-4P	1483	82.1	27.2	2.36	A	74.1	N	123	94.6	N	30.5	0.151	Alarma
2025-10-08 13:34:36	IMB3-TEFC-75kW-4P	1479	101.5	27.6	2.61	A	80.5	N	138.5	106.5	O	36.8	0.089	Alarma
2025-10-08 14:34:36	IMB3-TEFC-75kW-4P	1478	49.1	28	1.11	A	53.2	N	61.5	47.3	N	30.2	0.169	Alarma
2025-10-08 15:34:36	IMB3-TEFC-75kW-4P	1495	81.3	27.6	2.34	A	77.7	N	108.5	83.5	N	33	0.098	Alarma
2025-10-08 16:34:36	IMB3-TEFC-75kW-4P	1489	82.7	28.2	1.6	A	73.2	N	115.1	88.5	N	34.6	0.091	Alarma
2025-10-08 17:34:36	IMB3-TEFC-75kW-4P	1477	92.1	30.7	2.63	A	81.9	N	123.5	95	N	38.2	0.091	Alarma
2025-10-08 18:34:36	IMB3-TEFC-75kW-4P	1496	55.7	26.7	2.22	A	59.8	N	79.9	61.5	N	33.1	0.058	Alarma
2025-10-08 19:34:36	IMB3-TEFC-75kW-4P	1489	54.2	28.5	1.48	A	59.7	N	66.6	51.2	N	36.1	0.132	Alarma
2025-10-08 20:34:36	IMB3-TEFC-75kW-4P	1495	87.4	25.7	2.14	A	74.2	N	118.2	90.9	N	30	0.136	Alarma

2025-10-08 21:34:36	IMB3-TEFC-75kW-4P	1493	83.3	31	1.73	A	77.9	N	100.6	77.4	N	37	0.085	Alarma

Anexo N° 3. Código de simulación

```
# -*- coding: utf-8 -*-

import os
import json
from dataclasses import dataclass
from typing import Dict, List, Tuple
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


# ==========================
# 0) Utilidades generales
# ==========================

def info(msg: str) -> None:
    print(f"[INFO] {msg}")


# ==========================
# 1) Carga de datos
# ==========================

def load_data() -> Tuple[pd.DataFrame, str]:

    csv_path = "/content/measures1_v2.csv"
    xlsx_path = "salida_resumen_graficas.xlsx"

    if os.path.exists(csv_path):
        info(f"Cargando CSV: {csv_path}")
        df = pd.read_csv(csv_path)
        src = csv_path
    elif os.path.exists(xlsx_path):
        info(f"Cargando Excel: {xlsx_path}")
        xl = pd.ExcelFile(xlsx_path)
            sheet = "datos" if "datos" in xl.sheet_names else
xl.sheet_names[0]
        df = pd.read_excel(xlsx_path, sheet_name=sheet)
        src = f"{xlsx_path}::{sheet}"
    else:
             info("No se encontró CSV/Excel. Generando datos
sintéticos.")
        n = 6000
        rng = np.random.default_rng(1)
        df = pd.DataFrame({
            "motor_speed": rng.normal(3000, 900, n).clip(0),
```

```
                "torque": rng.normal(80, 35, n).clip(0),
                "stator_winding": rng.normal(60, 6, n).clip(20, 140),
                "stator_tooth": rng.normal(52, 6, n).clip(20, 140),
                "stator_yoke": rng.normal(40, 4, n).clip(10, 120),
                "pm": rng.normal(48, 5, n).clip(10, 140),
                "ambient": rng.normal(24, 3, n).clip(0, 60),
                "i_q": rng.normal(65, 22, n),
                "i_d": rng.normal(11, 6, n),
            })
            src = "(datos sintéticos)"

        # Asegurar columnas mínimas para el análisis
                                            need_cols                    =
["i_q","i_d","stator_winding","stator_tooth","stator_yoke","pm","motor_
speed","torque","ambient"]
        for c in need_cols:
            if c not in df.columns:
                df[c] = np.nan

        # Corriente vectorial total
                                    df["corriente_total"]                =
np.sqrt(np.square(df["i_q"].astype(float))                               +
np.square(df["i_d"].astype(float)))

        # Limitar tamaño para estabilidad de gráficos/tiempo
        df = df.head(5000).copy()

        return df, src


    # ==========================
    # 2) Configuración
    # ==========================

    @dataclass
    class EdgeNodeConfig:
        node_id: str
        sampling_hz: float
        window_s: float = 1.0
                        features:        Tuple[str,        ...]        =
("motor_speed","torque","stator_winding","stator_tooth","stator_yoke","
pm","ambient")
        pack_overhead_bytes: int = 32
        qos: int = 1             # 0,1,2
        target_latency_ms: float = 120.0


    @dataclass
    class LinkConfig:
```

```
        base_loss_pct: float
        mean_latency_ms: float
        std_latency_ms: float
        bandwidth_kbps: float
        jitter_ms: float


    @dataclass
    class LinkReliability:
        mtbf_s: float
        mttr_s: float


    # =========================
    # 3) Lógica de simulación
    # =========================

    def compute_edge_features(window: pd.DataFrame) -> Dict[str, float]:
        """
          Procesamiento local: estadísticas por señal + indicador térmico simple.
        """
        out: Dict[str, float] = {}
        for col in window.columns:
            if pd.api.types.is_numeric_dtype(window[col]):
                s = window[col].astype(float)
                out[f"{col}_mean"] = float(np.nanmean(s))
                out[f"{col}_std"]  = float(np.nanstd(s))
                out[f"{col}_min"]  = float(np.nanmin(s))
                out[f"{col}_max"]  = float(np.nanmax(s))
        if "stator_winding" in window.columns:
            sw = window["stator_winding"].astype(float)
             out["thermal_alert"] = float(np.mean(sw > 80.0))  # % de muestras >80°C
        return out


    def json_payload_size(payload: Dict[str, float], overhead: int) -> int:
        raw = json.dumps(payload, separators=(",", ":")).encode("utf-8")
        return len(raw) + overhead


    _rng = np.random.default_rng(42)

    def qos_loss(base_loss_pct: float, qos: int) -> float:
        """
```

```
        Reducción de pérdida por QoS.
        """
        if qos <= 0:
            return base_loss_pct
        if qos == 1:
            return base_loss_pct * 0.35
        return base_loss_pct * 0.10   # QoS 2


    def sample_latency(mean_ms: float, std_ms: float, jitter_ms:
float, qos: int) -> float:
        """
        Latencia ~ N(mean, std) + jitter +/- + overhead por QoS.
        """
        base = _rng.normal(mean_ms, std_ms)
        jit  = _rng.uniform(-jitter_ms, jitter_ms)
        extra = 0.0 if qos == 0 else (5.0 if qos == 1 else 15.0)
        return max(0.0, base + jit + extra)


    def make_windows(subdf: pd.DataFrame, sampling_hz: float,
window_s: float, total_s: int) -> List[pd.DataFrame]:
        """
         Construye ventanas consecutivas con tamaño ~ sampling_hz *
window_s.
        Si se agotan las filas, recircula desde el inicio para cubrir
total_s.
        """
        win_len = max(1, int(round(sampling_hz * window_s)))
        out: List[pd.DataFrame] = []
        i = 0
        data = subdf.reset_index(drop=True)
        for _ in range(total_s):
            j = i + win_len
            if j <= len(data):
                w = data.iloc[i:j]
                i = j
            else:
                rem = j - len(data)
                    w = pd.concat([data.iloc[i:], data.iloc[:rem]],
ignore_index=True)
                i = rem
            out.append(w)
        return out


    def simulate_link_states(mtbf_s: float, mttr_s: float,
duration_s: int) -> List[Tuple[float, float, bool]]:
        """
```

```
        Retorna intervalos (t0, t1, up_flag). Inicia UP.
        """
        t = 0.0
        up = True
        ivals: List[Tuple[float, float, bool]] = []
        while t < duration_s:
            if up:
                up_time = _rng.exponential(mtbf_s)
                t1 = min(duration_s, t + up_time)
                ivals.append((t, t1, True))
                t = t1
                up = False
            else:
                down_time = _rng.exponential(mttr_s)
                t1 = min(duration_s, t + down_time)
                ivals.append((t, t1, False))
                t = t1
                up = True
        return ivals


    def link_state_at(intervals: List[Tuple[float, float, bool]], t:
float) -> bool:
        for t0, t1, st in intervals:
            if t0 <= t < t1:
                return st
        return intervals[-1][2]


    def summarize_reliability(intervals: List[Tuple[float, float,
bool]], duration_s: int) -> Tuple[float, float, float, int, int]:
        up_total = sum((t1 - t0) for t0, t1, st in intervals if st)
        down_total = duration_s - up_total
          fails = sum(1 for k in range(1, len(intervals)) if
intervals[k-1][2] and not intervals[k][2])
         repairs = sum(1 for k in range(1, len(intervals)) if (not
intervals[k-1][2]) and intervals[k][2])
        mean_up = up_total / max(1, fails) if fails > 0 else up_total
         mean_down = down_total / max(1, repairs) if repairs > 0 else
(down_total if down_total > 0 else 0.0)
         availability = 100.0 * up_total / duration_s if duration_s >
0 else 0.0
        return availability, mean_up, mean_down, fails, repairs


    def apply_ntp_drift_and_sync(total_s: int, sync_period_s: int,
drift_ppm: float = 30.0) -> Tuple[List[float], List[float], int]:
        """
```

```
            Deriva de reloj en ms (random walk) + sincronización
periódica cada sync_period_s.
                Devuelve drift_before, drift_after y número de
sincronizaciones.
            """
            drift_ms = _rng.uniform(-sync_period_s * drift_ppm * 1e-3,
                                    sync_period_s * drift_ppm * 1e-3)
            before: List[float] = []
            after:  List[float] = []
            syncs = 0
            for t in range(total_s):
                drift_ms += _rng.normal(0.0, 0.5)  # random walk leve
                before.append(drift_ms)
                if (t + 1) % sync_period_s == 0:
                     sync_error = _rng.normal(0.0, 1.0)  # error residual
tras sync
                    drift_ms = drift_ms * 0.2 + sync_error
                    syncs += 1
                after.append(drift_ms)
            return before, after, syncs


        # ==========================
        # 4) Ejecución multi-nodo
        # ==========================

        def run_simulation(df: pd.DataFrame,
                           duration_s: int = 120,
                                        sync_period_s:  int  =  10)  ->
Tuple[pd.DataFrame, pd.DataFrame, pd.DataFrame, List[float]]:
            """
            Ejecuta la simulación para 3 nodos con config por defecto.
            Retorna:
              - metrics_df: indicadores por nodo + total
              - link_df: intervalos up/down
              - drift_df: deriva previa y posterior a sync
              - all_latencies: lista consolidada de latencias recibidas
            """

            # Config nodos (puedes ajustar aquí fácilmente)
            edge_nodes = [
                      EdgeNodeConfig(node_id="edge-1",   sampling_hz=50.0,
qos=1),
                      EdgeNodeConfig(node_id="edge-2",   sampling_hz=75.0,
qos=1),
                     EdgeNodeConfig(node_id="edge-3",   sampling_hz=100.0,
qos=2),
            ]
            links = [
```

```
                    LinkConfig(base_loss_pct=0.8,   mean_latency_ms=70,
std_latency_ms=20, bandwidth_kbps=256, jitter_ms=10),
                    LinkConfig(base_loss_pct=1.2,   mean_latency_ms=85,
std_latency_ms=25, bandwidth_kbps=224, jitter_ms=10),
                    LinkConfig(base_loss_pct=2.0,   mean_latency_ms=95,
std_latency_ms=30, bandwidth_kbps=192, jitter_ms=12),
          ]
          reliab = [
              LinkReliability(mtbf_s=90,  mttr_s=12),
              LinkReliability(mtbf_s=120, mttr_s=15),
              LinkReliability(mtbf_s=150, mttr_s=18),
          ]

          # Distribuir datos entre nodos (barajado)
                  node_dfs     =     np.array_split(df.sample(frac=1.0,
random_state=7).reset_index(drop=True), len(edge_nodes))

          metrics_rows = []
          link_rows = []
          drift_rows = []
          all_latencies: List[float] = []

           for cfg, link, rel, subdf in zip(edge_nodes, links, reliab,
node_dfs):
              # Ventanas de procesamiento (1 s)
                    windows   =   make_windows(subdf,   cfg.sampling_hz,
cfg.window_s, duration_s)

                  # Estados de enlace y métricas de confiabilidad
observadas
               intervals = simulate_link_states(rel.mtbf_s, rel.mttr_s,
duration_s)
                    link_avail_pct,   obs_mtbf,   obs_mttr,   _,   _   =
summarize_reliability(intervals, duration_s)

              # Deriva NTP y sincronización
                 pre,  post,  _  =  apply_ntp_drift_and_sync(duration_s,
sync_period_s, drift_ppm=30.0)
              mean_drift = float(np.mean(np.abs(post)))
              p95_drift  = float(np.percentile(np.abs(post), 95))

              # Transmisión
               loss_effective = qos_loss(link.base_loss_pct, cfg.qos) /
100.0
              msgs_sent = 0
              msgs_recv = 0
              bits_total = 0
              latencies: List[float] = []
```

```
            for sec, w in enumerate(windows):
                # Columnas disponibles del nodo
                    cols_avail = [c for c in cfg.features if c in
w.columns]
                if not cols_avail:
                            cols_avail = [c for c in w.columns if
pd.api.types.is_numeric_dtype(w[c])]
                ww = w[cols_avail].copy()

                # Features locales
                features = compute_edge_features(ww)
                features["node_id"]  = cfg.node_id
                features["sec"]      = sec
                features["drift_ms"] = pre[sec]

                # Tamaño y conteos
                            size_b   =   json_payload_size(features,
cfg.pack_overhead_bytes)
                bits_total += size_b * 8
                msgs_sent += 1

                # Estado de enlace y pérdidas
                 if not link_state_at(intervals, sec):  # down => se
pierde
                    pass
                else:
                             if  np.random.default_rng().random()  <
loss_effective:
                        pass
                    else:
                        # Entrega exitosa: latencia
                           lat = sample_latency(link.mean_latency_ms,
link.std_latency_ms, link.jitter_ms, cfg.qos)
                        latencies.append(lat)
                        msgs_recv += 1

            # Indicadores por nodo
             bandwidth_kbps = (bits_total / duration_s) / 1000.0 if
duration_s > 0 else 0.0
              loss_pct = 100.0 * (msgs_sent - msgs_recv) / max(1,
msgs_sent)
            thr = msgs_recv / duration_s
             med_lat = float(np.median(latencies)) if latencies else
0.0
                 p95_lat  =  float(np.percentile(latencies,  95))  if
latencies else 0.0
```

```
                ok_msgs = sum(1 for x in latencies if x <=
cfg.target_latency_ms)
            availability = 100.0 * (ok_msgs / max(1, msgs_sent))

            metrics_rows.append({
                "node_id": cfg.node_id,
                "sampling_hz": cfg.sampling_hz,
                "msgs_sent": msgs_sent,
                "msgs_recv": msgs_recv,
                "loss_pct": round(loss_pct, 2),
                "throughput_msg_s": round(thr, 3),
                "median_latency_ms": round(med_lat, 2),
                "p95_latency_ms": round(p95_lat, 2),
                "bandwidth_kbps": round(bandwidth_kbps, 2),
                "availability_pct": round(availability, 2),
                "link_availability_pct": round(link_avail_pct, 2),
                "observed_mtbf_s": round(obs_mtbf, 2),
                "observed_mttr_s": round(obs_mttr, 2),
                "mean_drift_ms": round(mean_drift, 2),
                "p95_drift_ms": round(p95_drift, 2),
            })
            all_latencies += latencies

            # Guardar intervalos UP/DOWN y drift
            for (t0, t1, st) in intervals:
                link_rows.append({"node_id": cfg.node_id, "t0_s": t0,
"t1_s": t1, "up": int(st)})
            for sec, (d0, d1) in enumerate(zip(pre, post)):
                drift_rows.append({"node_id": cfg.node_id, "sec":
sec, "drift_before_ms": d0, "drift_after_ms": d1})

        metrics_df = pd.DataFrame(metrics_rows)

        # Fila total consolidada
        totals = {
            "node_id": "TOTAL",
            "sampling_hz": round(metrics_df["sampling_hz"].mean(),
3),
            "msgs_sent": int(metrics_df["msgs_sent"].sum()),
            "msgs_recv": int(metrics_df["msgs_recv"].sum()),
            "loss_pct": round(100.0 * (metrics_df["msgs_sent"].sum()
- metrics_df["msgs_recv"].sum())
                                                    /   max(1,
metrics_df["msgs_sent"].sum()), 2),
            "throughput_msg_s": round(metrics_df["msgs_recv"].sum() /
duration_s, 3),
                                                "median_latency_ms":
round(float(np.median(all_latencies)) if all_latencies else 0.0, 2),
```

```
                                                                "p95_latency_ms":
round(float(np.percentile(all_latencies,  95))  if  all_latencies  else
0.0, 2),
                                                                "bandwidth_kbps":
round(metrics_df["bandwidth_kbps"].sum(), 2),
                "availability_pct": round((metrics_df["availability_pct"]
* metrics_df["msgs_sent"]).sum()
                                                                        /  max(1,
metrics_df["msgs_sent"].sum()), 2),
                                                        "link_availability_pct":
round((metrics_df["link_availability_pct"]                                  *
metrics_df["msgs_sent"]).sum()
                                                                        /  max(1,
metrics_df["msgs_sent"].sum()), 2),
                "observed_mtbf_s": round((metrics_df["observed_mtbf_s"] *
metrics_df["msgs_sent"]).sum()
                                                                        /  max(1,
metrics_df["msgs_sent"].sum()), 2),
                "observed_mttr_s": round((metrics_df["observed_mttr_s"] *
metrics_df["msgs_sent"]).sum()
                                                                        /  max(1,
metrics_df["msgs_sent"].sum()), 2),
                  "mean_drift_ms":  round((metrics_df["mean_drift_ms"]  *
metrics_df["msgs_sent"]).sum()
                                                                        /  max(1,
metrics_df["msgs_sent"].sum()), 2),
                   "p95_drift_ms":  round((metrics_df["p95_drift_ms"]  *
metrics_df["msgs_sent"]).sum()
                                                                        /  max(1,
metrics_df["msgs_sent"].sum()), 2),
            }
             metrics_df  =  pd.concat([metrics_df,  pd.DataFrame([totals])],
ignore_index=True)

            link_df  = pd.DataFrame(link_rows)
            drift_df = pd.DataFrame(drift_rows)
            return metrics_df, link_df, drift_df, all_latencies


        # ==========================
        # 5) Gráficas
        # ==========================

        def  plot_latency_hist(all_latencies:  List[float],  path:  str)  ->
None:
            plt.figure(figsize=(8, 5))
            plt.hist(all_latencies, bins=25)
            plt.xlabel("Latencia de transmisión (ms)")
            plt.ylabel("Frecuencia")
```

```
        plt.title("Distribución de Latencia (todos los nodos)")
        plt.grid(True)
        plt.tight_layout()
        plt.savefig(path, dpi=140)
        plt.close()


    def plot_throughput_bar(metrics_df: pd.DataFrame, path: str) ->
None:
        plt.figure(figsize=(8, 5))
        # Excluir fila TOTAL
        mask = metrics_df["node_id"] != "TOTAL"
         plt.bar(metrics_df.loc[mask, "node_id"], metrics_df.loc[mask,
"throughput_msg_s"])
        plt.xlabel("Nodo Edge")
        plt.ylabel("Throughput (msg/s)")
        plt.title("Throughput por nodo")
        plt.grid(axis="y")
        plt.tight_layout()
        plt.savefig(path, dpi=140)
        plt.close()


    def plot_heatmap_current_torque_temp(df: pd.DataFrame, path: str)
-> None:
            valid = df.dropna(subset=["corriente_total", "torque",
"stator_winding"])
        if valid.empty:
            print("[WARN] No hay datos válidos para heatmap.")
            return
        x = valid["corriente_total"].values
        y = valid["torque"].values
        z = valid["stator_winding"].values

        xbins, ybins = 30, 30
        x_edges = np.linspace(x.min(), x.max(), xbins + 1)
        y_edges = np.linspace(y.min(), y.max(), ybins + 1)

        sum_temp = np.zeros((ybins, xbins))
        count = np.zeros((ybins, xbins))

        xi = np.clip(np.digitize(x, x_edges) - 1, 0, xbins - 1)
        yi = np.clip(np.digitize(y, y_edges) - 1, 0, ybins - 1)
        for a, b, t in zip(xi, yi, z):
            sum_temp[b, a] += t
            count[b, a] += 1

                avg_temp = np.divide(sum_temp, count,
out=np.zeros_like(sum_temp), where=count > 0)
```

```
          plt.figure(figsize=(8, 6))
          plt.imshow(avg_temp, origin="lower", aspect="auto",
                           extent=[x_edges[0],  x_edges[-1],  y_edges[0],
y_edges[-1]])
          plt.colorbar(label="Temp. devanado promedio (°C)")
          plt.xlabel("Corriente total (A)")
          plt.ylabel("Torque (Nm)")
             plt.title("Heatmap  corriente  vs  torque  →  temperatura
promedio")
          plt.tight_layout()
          plt.savefig(path, dpi=140)
          plt.close()


      def  plot_current_speed_curves_by_thermal_zone(df:  pd.DataFrame,
path: str) -> None:
           valid  =  df.dropna(subset=["corriente_total",  "motor_speed",
"stator_winding"])
          if valid.empty:
              print("[WARN] No hay datos válidos para curvas.")
              return

          zones = [
              ("Normal", valid["stator_winding"] < 70.0),
                 ("Observación",  (valid["stator_winding"]  >=  70.0)  &
(valid["stator_winding"] < 85.0)),
              ("Alarma", valid["stator_winding"] >= 85.0),
          ]
          v = valid["motor_speed"].values
          i_tot = valid["corriente_total"].values
          v_edges = np.linspace(np.nanmin(v), np.nanmax(v), 21)
          v_mid = 0.5 * (v_edges[:-1] + v_edges[1:])

          plt.figure(figsize=(9, 6))
          for name, mask in zones:
              v_sel = v[mask.values]
              i_sel = i_tot[mask.values]
              medians = []
              for k in range(len(v_edges) - 1):
                  m = (v_sel >= v_edges[k]) & (v_sel < v_edges[k + 1])
                        medians.append(float(np.nanmedian(i_sel[m]))  if
m.any() else np.nan)
              medians = np.array(medians, dtype=float)
              plt.plot(v_mid, medians, label=name)

          plt.xlabel("Velocidad (RPM)")
          plt.ylabel("Corriente total (A)")
```

```
          plt.title("Curvas de operación – Corriente vs Velocidad por
zonas térmicas")
          plt.legend(title="Zona térmica")
          plt.grid(True)
          plt.tight_layout()
          plt.savefig(path, dpi=140)
          plt.close()


      # =========================
      # 6) Exportación a Excel
      # =========================

      def export_excel(metrics_df: pd.DataFrame, link_df: pd.DataFrame,
drift_df: pd.DataFrame, path: str) -> None:
          with pd.ExcelWriter(path, engine="xlsxwriter") as w:
                      metrics_df.to_excel(w,   sheet_name="indicadores",
index=False)
                         link_df.to_excel(w,   sheet_name="link_states",
index=False)
              drift_df.to_excel(w, sheet_name="ntp_drift", index=False)


      # =========================
      # 7) Main
      # =========================

      def main():
          df, src = load_data()
          info(f"Fuente de datos: {src}")

          DURATION_S = 120
          SYNC_PERIOD_S = 10

          info("Ejecutando simulación…")
               metrics_df,   link_df,   drift_df,   all_latencies   =
run_simulation(
              df, duration_s=DURATION_S, sync_period_s=SYNC_PERIOD_S
          )

          # Exportar Excel con indicadores y trazabilidad
          xlsx_out = "indicadores_edge_mqtt_ext.xlsx"
          export_excel(metrics_df, link_df, drift_df, xlsx_out)
          info(f"Excel exportado: {xlsx_out}")

          # Gráficas
          plot_latency_hist(all_latencies, "latencia_hist_ext.png")
          plot_throughput_bar(metrics_df, "throughput_barras_ext.png")
```

```
                                    plot_heatmap_current_torque_temp(df,
"heatmap_corriente_torque_temp.png")
                            plot_current_speed_curves_by_thermal_zone(df,
"curvas_corriente_velocidad_zonas.png")
                info("Gráficas    generadas:    latencia_hist_ext.png,
throughput_barras_ext.png, "
                                     "heatmap_corriente_torque_temp.png,
curvas_corriente_velocidad_zonas.png")

          # Resumen corto por consola
          print("\n=== RESUMEN (TOTAL) ===")
                  total     =     metrics_df[metrics_df["node_id"]     ==
"TOTAL"].iloc[0].to_dict()
          for k, v in total.items():
              print(f"{k}: {v}")


      if __name__ == "__main__":
          main()

      # -*- coding: utf-8 -*-

      from __future__ import annotations
      from dataclasses import dataclass, field, asdict
      from typing import Dict, List, Tuple, Optional, Callable
      import numpy as np
      import pandas as pd

      # =========================
      # 0) Parámetros de salida
      # =========================
      OUT_XLSX = "edge_acq_arquitectura_4_2.xlsx"
      MAX_TIMELINE_PER_NODE  =  2000         #  filas  máx  por  nodo  en
Timeline
      MAX_FEATURE_ROWS  =  None             # None = todas; o un entero
para acotar tamaño


      # =========================
      # 1) Utilidades de tiempo
      # =========================

      @dataclass
      class ClockModel:
          """Modelo simple de reloj (drift + wander)."""
          drift_ppm: float = 0.0
          wander_ms: float = 0.0
          _offset_ms: float = 0.0
```

```
        def tick(self, seconds: float) -> None:
            # deriva estática (ppm -> ms/s)
            self._offset_ms += (self.drift_ppm * 1e-3) * seconds
            # random walk
            if self.wander_ms > 0.0:
                        self._offset_ms   +=   np.random.normal(0.0,
self.wander_ms)

        def now(self, t_wall_s: float) -> float:
            """Convierte tiempo de pared (s) a tiempo local (s)."""
            return t_wall_s + self._offset_ms / 1000.0


    # ==========================
    # 2) Sensores y muestreo
    # ==========================

    @dataclass
    class Sensor:
        """
        Sensor genérico (simulado) con generador f(t_local) -> valor.
        """
        name: str
        generator: Callable[[float], float]
        noise_std: float = 0.0
        unit: str = ""

        def read(self, t_local_s: float) -> float:
            val = float(self.generator(t_local_s))
            if self.noise_std > 0:
                val += np.random.normal(0.0, self.noise_std)
            return val


    @dataclass
    class Sampler:
        """
        Planificador de muestreo del nodo.
        """
        fs_hz: float
        max_jitter_ms: float = 0.0
        clock: ClockModel = field(default_factory=ClockModel)

         def schedule(self, t0_wall_s: float, duration_s: float) ->
np.ndarray:
             """Devuelve tiempos de PARED (s) en los que muestrear
(incluye jitter)."""
            if self.fs_hz <= 0:
                return np.array([])
```

```
            n = int(np.floor(duration_s * self.fs_hz))
            if n == 0:
                return np.array([])
            dt = 1.0 / self.fs_hz
            t_wall = t0_wall_s + np.arange(n) * dt
            if self.max_jitter_ms > 0:
                    jitter  =  np.random.uniform(-self.max_jitter_ms,
self.max_jitter_ms, size=n) / 1000.0
                t_wall = t_wall + jitter
            return t_wall


    # =========================
    # 3) Buffer circular
    # =========================

    class RingBuffer:
        """
          Buffer  circular  de  dicts  {sensor:  valor,  'ts_local',
'ts_wall'} con conteo de overflow.
        """
        def __init__(self, capacity: int):
            self.capacity = int(capacity)
            self._buf: List[Optional[dict]] = [None] * self.capacity
            self._w = 0
            self._count = 0
            self.dropped = 0

        def push(self, item: dict):
            if self._count < self.capacity:
                self._buf[self._w] = item
                self._w = (self._w + 1) % self.capacity
                self._count += 1
            else:
                # overflow: sobreescribe y cuenta pérdida
                self._buf[self._w] = item
                self._w = (self._w + 1) % self.capacity
                self.dropped += 1

        def drain_window(self, window_samples: int) -> List[dict]:
            if self._count < window_samples:
                return []
            start = (self._w - self._count) % self.capacity
            out = []
            for _ in range(window_samples):
                item = self._buf[start]
                out.append(item)
                self._buf[start] = None
                start = (start + 1) % self.capacity
```

```
                self._count -= 1
            return out

        def snapshot(self) -> List[dict]:
            if self._count == 0:
                return []
            start = (self._w - self._count) % self.capacity
            out = []
            for i in range(self._count):
                idx = (start + i) % self.capacity
                out.append(self._buf[idx])
            return out

        def __len__(self):
            return self._count


    # ==========================
    # 4) Nodo Edge (adquisición)
    # ==========================

    @dataclass
    class EdgeNode:
        node_id: str
        sensors: List[Sensor]
        sampler: Sampler
        buffer_capacity: int = 5000
        window_s: float = 1.0

        # internos
        _buf: RingBuffer = field(init=False)
        _samples_total: int = 0
        _samples_overflow: int = 0
        _ts_local: List[float] = field(default_factory=list)
        _ts_wall: List[float] = field(default_factory=list)
        _windows_out: int = 0

        def __post_init__(self):
            self._buf = RingBuffer(self.buffer_capacity)

         def acquire(self, t0_wall_s: float, duration_s: float) ->
None:
            """Simula adquisición durante 'duration_s' segundos y
llena el buffer."""
            t_wall = self.sampler.schedule(t0_wall_s, duration_s)
            if t_wall.size == 0:
                return
            for tw in t_wall:
                # avanzar reloj local (tick ~ 1/fs)
```

```
                            self.sampler.clock.tick(1.0 / max(1e-9,
self.sampler.fs_hz))
                t_local = self.sampler.clock.now(tw)
                row = {"ts_wall": tw, "ts_local": t_local}
                for s in self.sensors:
                    row[s.name] = s.read(t_local)
                before = self._buf.dropped
                self._buf.push(row)
                self._samples_total += 1
                     self._samples_overflow += (self._buf.dropped -
before)
                self._ts_wall.append(tw)
                self._ts_local.append(t_local)

        def make_windows(self) -> List[pd.DataFrame]:
                """Convierte el buffer en ventanas de tamaño fijo
(consume el buffer)."""
                win_samples = max(1, int(round(self.sampler.fs_hz *
self.window_s)))
            windows = []
            while True:
                chunk = self._buf.drain_window(win_samples)
                if not chunk:
                    break
                windows.append(pd.DataFrame(chunk))
                self._windows_out += 1
            return windows

        @staticmethod
         def compute_features(window_df: pd.DataFrame) -> Dict[str,
float]:
             """Features estadísticas por ventana (media, std, min,
max) y calidad temporal."""
            feats: Dict[str, float] = {}
               cols = [c for c in window_df.columns if c not in
("ts_wall", "ts_local")]
            for c in cols:
                s = window_df[c].astype(float)
                feats[f"{c}_mean"] = float(np.nanmean(s))
                feats[f"{c}_std"]  = float(np.nanstd(s))
                feats[f"{c}_min"]  = float(np.nanmin(s))
                feats[f"{c}_max"]  = float(np.nanmax(s))
            tl = window_df["ts_local"].values
            if len(tl) >= 2:
                dt = np.diff(tl) * 1000.0  # ms
                feats["dt_median_ms"] = float(np.median(dt))
                feats["dt_p95_ms"]    = float(np.percentile(dt, 95))
            return feats
```

```
        def acquisition_metrics(self) -> Dict[str, float]:
            """Indicadores de adquisición por nodo."""
            if len(self._ts_wall) <= 1:
                return {
                                    "node_id":  self.node_id,  "samples":
self._samples_total,
                    "loss_overflow_pct": 0.0, "fs_effective_hz": 0.0,
                    "sampling_jitter_ms_p95": 0.0, "coverage_s": 0.0,
                        "node_availability_pct":  0.0,  "windows_out":
self._windows_out,
                    "windows_rate_per_s": 0.0,
                }
            elapsed = self._ts_wall[-1] - self._ts_wall[0]
            fs_eff = (len(self._ts_wall) - 1) / max(1e-9, elapsed)
             dt_local = np.diff(np.array(self._ts_local)) * 1000.0  #
ms
                jitter_p95  =  float(np.percentile(np.abs(dt_local  -
np.median(dt_local)), 95))
                loss_pct  =  100.0  *  self._samples_overflow  /  max(1,
self._samples_total)
            coverage_s = float(elapsed)
                  availability  =  100.0  if  fs_eff  >=  0.8  *
self.sampler.fs_hz else 0.0
            win_rate = self._windows_out / max(1e-9, coverage_s)
            return {
                "node_id": self.node_id,
                "samples": int(self._samples_total),
                "loss_overflow_pct": round(loss_pct, 3),
                "fs_effective_hz": round(fs_eff, 3),
                "sampling_jitter_ms_p95": round(jitter_p95, 3),
                "coverage_s": round(coverage_s, 3),
                "node_availability_pct": round(availability, 2),
                "windows_out": int(self._windows_out),
                "windows_rate_per_s": round(win_rate, 3),
            }


    # ==========================
    # 5) Generadores de señales
    # ==========================

    def gen_speed(t: float) -> float:
            return  1500  +  100*np.sin(2*np.pi*0.02*t)  +
10*np.sin(2*np.pi*1.2*t)

    def gen_torque(t: float) -> float:
        return 100 + 20*np.sin(2*np.pi*0.1*t) + 5*np.random.randn()
```

```
        def gen_iq(t: float) -> float:
            return 60 + 10*np.sin(2*np.pi*0.5*t)

        def gen_id(t: float) -> float:
            return 8 + 2*np.sin(2*np.pi*0.25*t + 0.6)

        def gen_temp_w(t: float) -> float:
            return 65 + 0.4*np.sin(2*np.pi*0.005*t)

        def gen_temp_b(t: float) -> float:
            return 40 + 0.3*np.sin(2*np.pi*0.007*t + 1.1)

        def gen_ambient(t: float) -> float:
            return 25 + 0.2*np.sin(2*np.pi*0.002*t + 0.5)


        # ==========================
        # 6) Construcción de nodos
        # ==========================

        def build_default_nodes() -> List[EdgeNode]:
             """Crea 3 nodos edge con diferentes perfiles de muestreo,
reloj y sensores."""
            n1 = EdgeNode(
                node_id="edge-A",
                sensors=[
                      Sensor("motor_speed_rpm", gen_speed, noise_std=0.5,
unit="rpm"),
                         Sensor("torque_Nm", gen_torque, noise_std=0.8,
unit="Nm"),
                    Sensor("i_q_A", gen_iq, noise_std=0.2, unit="A"),
                    Sensor("i_d_A", gen_id, noise_std=0.1, unit="A"),
                     Sensor("temp_winding_C", gen_temp_w, noise_std=0.05,
unit="°C"),
                     Sensor("temp_bearing_C", gen_temp_b, noise_std=0.05,
unit="°C"),
                        Sensor("ambient_C", gen_ambient, noise_std=0.02,
unit="°C"),
                ],
                         sampler=Sampler(fs_hz=100.0,   max_jitter_ms=0.2,
clock=ClockModel(drift_ppm=3, wander_ms=0.05)),
                buffer_capacity=8000,
                window_s=1.0,
            )

            n2 = EdgeNode(
                node_id="edge-B",
                sensors=[
```

```
                    Sensor("motor_speed_rpm", gen_speed, noise_std=0.7,
unit="rpm"),
                        Sensor("torque_Nm", gen_torque, noise_std=1.0,
unit="Nm"),
                Sensor("i_q_A", gen_iq, noise_std=0.3, unit="A"),
                Sensor("i_d_A", gen_id, noise_std=0.15, unit="A"),
                 Sensor("temp_winding_C", gen_temp_w, noise_std=0.05,
unit="°C"),
                    Sensor("ambient_C", gen_ambient, noise_std=0.02,
unit="°C"),
            ],
                        sampler=Sampler(fs_hz=50.0,   max_jitter_ms=0.6,
clock=ClockModel(drift_ppm=8, wander_ms=0.1)),
            buffer_capacity=5000,
            window_s=1.0,
        )

        n3 = EdgeNode(
            node_id="edge-C",
            sensors=[
                  Sensor("motor_speed_rpm", gen_speed, noise_std=0.9,
unit="rpm"),
                      Sensor("torque_Nm", gen_torque, noise_std=1.2,
unit="Nm"),
                Sensor("i_q_A", gen_iq, noise_std=0.35, unit="A"),
                Sensor("i_d_A", gen_id, noise_std=0.2, unit="A"),
                 Sensor("temp_bearing_C", gen_temp_b, noise_std=0.05,
unit="°C"),
                    Sensor("ambient_C", gen_ambient, noise_std=0.02,
unit="°C"),
            ],
                        sampler=Sampler(fs_hz=25.0,   max_jitter_ms=1.5,
clock=ClockModel(drift_ppm=20, wander_ms=0.2)),
            buffer_capacity=2500,
            window_s=2.0,
        )
        return [n1, n2, n3]


    # =========================
    # 7) Orquestación 4.2
    # =========================

    def run_acquisition_to_xlsx(duration_s: float = 30.0,
                                out_xlsx: str = OUT_XLSX,
                                        max_timeline_per_node: int =
MAX_TIMELINE_PER_NODE,
                                    max_feature_rows: Optional[int] =
MAX_FEATURE_ROWS) -> str:
```

```
        """
        Ejecuta la arquitectura de adquisición distribuida y exporta
todo a un XLSX.
        Devuelve la ruta del archivo generado.
        """
        nodes = build_default_nodes()
        t0 = 0.0  # tiempo de pared base arbitrario

        # 1) Adquisición
        for n in nodes:
            n.acquire(t0_wall_s=t0, duration_s=duration_s)

        # 2) Ventanas + features
        feat_rows = []
        for n in nodes:
            wins = n.make_windows()
            for k, w in enumerate(wins):
                f = n.compute_features(w)
                f["node_id"] = n.node_id
                f["window_idx"] = k
                f["window_s"] = n.window_s
                feat_rows.append(f)
        features_df = pd.DataFrame(feat_rows)
          if max_feature_rows is not None and len(features_df) >
max_feature_rows:
            features_df = features_df.head(max_feature_rows).copy()

        # 3) Métricas por nodo
         metrics_df = pd.DataFrame([n.acquisition_metrics() for n in
nodes])

          # 4) Config de nodos/sensores (para dejar arquitectura
explícita)
        cfg_rows = []
        sens_rows = []
        for n in nodes:
            cfg_rows.append({
                "node_id": n.node_id,
                "fs_objetivo_hz": n.sampler.fs_hz,
                "window_s": n.window_s,
                "buffer_capacity": n.buffer_capacity,
                "max_jitter_ms": n.sampler.max_jitter_ms,
                "clock_drift_ppm": n.sampler.clock.drift_ppm,
                "clock_wander_ms": n.sampler.clock.wander_ms,
            })
            for s in n.sensors:
                sens_rows.append({
```

```
                    "node_id": n.node_id, "sensor": s.name, "unidad":
s.unit, "noise_std": s.noise_std
                })
        config_df = pd.DataFrame(cfg_rows)
        sensors_df = pd.DataFrame(sens_rows)

        # 5) Timeline (muestra para inspección)
        tl_rows = []
        for n in nodes:
            # ts ya utilizados (hasta límite)
                    for    tsw,    tsl    in    list(zip(n._ts_wall,
n._ts_local))[:max_timeline_per_node]:
                tl_rows.append({"node_id": n.node_id, "ts_wall": tsw,
"ts_local": tsl})
            # instantánea remanente del buffer (si algo quedó)
            snap = n._buf.snapshot()[: max(0, max_timeline_per_node -
len(tl_rows))]
            for s in snap:
                    tl_rows.append({"node_id":  n.node_id,  "ts_wall":
s["ts_wall"], "ts_local": s["ts_local"]})
        timeline_df = pd.DataFrame(tl_rows)

        # 6) Hoja de arquitectura (texto + diagrama ASCII)
        arquitectura_texto = [
            "4.2 - Diseño de arquitectura embebida distribuida para
adquisición de datos",
            "",
            "Arquitectura (solo adquisición local):",
               "   [Sensores]  ->  [Sampler  (Hz,  jitter,  reloj)]  ->
[RingBuffer (FIFO circular)] ->",
              "   [Ventanas  (window_s)]  ->  [Features  locales  (media,
std, min, max, dt_median/p95)]",
            "",
            "No incluye transmisión (MQTT) en esta etapa.",
            "",
            "Nodos y responsabilidades:",
             "  - edge-A: 100 Hz, reloj estable; conjunto completo de
sensores (corriente, torque, térmico).",
                "   -  edge-B:  50  Hz,  jitter  moderado;  subset  de
sensores.",
             "  - edge-C: 25 Hz, jitter alto; ventana de 2 s (mejora
robustez con baja tasa).",
            "",
              "Indicadores  clave  de  adquisición  por  nodo  (ver  hoja
02_metrics):",
                  "    -  fs_effective_hz:  Hz  alcanzados  (no  solo
configurados).",
            "  - sampling_jitter_ms_p95: estabilidad temporal.",
```

```
                "   - loss_overflow_pct: pérdida por overflow del
buffer.",
            "  - coverage_s: cobertura temporal efectiva.",
                "   - node_availability_pct: 100% si fs_effective >=
0.8*fs_objetivo.",
                "   - windows_out y windows_rate_per_s: rendimiento de
features/ventanas.",
        ]
                arquitectura_df     =     pd.DataFrame({"arquitectura":
arquitectura_texto})

        # 7) Diccionario de datos / definiciones
        diccionario = [
            ("node_id", "Identificador del nodo edge."),
              ("fs_objetivo_hz", "Frecuencia de muestreo deseada del
nodo."),
                 ("fs_effective_hz", "Frecuencia efectiva alcanzada
(Hz)."),
              ("max_jitter_ms", "Jitter máximo (uniforme) aplicado a
los instantes de muestreo."),
                ("sampling_jitter_ms_p95", "Percentil 95 del jitter
relativo (ms)."),
              ("clock_drift_ppm", "Deriva estática del reloj del nodo
(ppm)."),
                ("clock_wander_ms", "Random walk de reloj por tick
(ms)."),
              ("buffer_capacity", "Capacidad del buffer circular del
nodo (muestras)."),
               ("loss_overflow_pct", "Porcentaje de muestras perdidas
por overflow del buffer."),
              ("window_s", "Duración de cada ventana de procesamiento
local (s)."),
            ("windows_out", "Número total de ventanas generadas."),
            ("windows_rate_per_s", "Ventanas por segundo."),
            ("coverage_s", "Cobertura temporal adquirida (s)."),
              ("dt_median_ms", "Mediana del espaciado temporal entre
muestras en la ventana (ms)."),
                ("dt_p95_ms", "Percentil 95 del espaciado temporal
(ms)."),
        ]
            dicc_df   =   pd.DataFrame(diccionario,   columns=["campo",
"definición"])

            #   8)   Exportar   a   Excel   (engine=openpyxl   para   mejor
compatibilidad)
        with pd.ExcelWriter(out_xlsx, engine="openpyxl") as w:
            arquitectura_df.to_excel(w, sheet_name="00_arquitectura",
index=False)
```

```
            # Unimos config y sensores en una sola hoja (dos tablas)
                 config_df.to_excel(w,  sheet_name="01_config_nodos",
index=False, startrow=0)
             startrow = len(config_df) + 3
                 sensors_df.to_excel(w,  sheet_name="01_config_nodos",
index=False, startrow=startrow)
                     metrics_df.to_excel(w,   sheet_name="02_metrics",
index=False)
                   features_df.to_excel(w,   sheet_name="03_features",
index=False)
              timeline_df.to_excel(w, sheet_name="04_timeline_muestra",
index=False)
                    dicc_df.to_excel(w,   sheet_name="05_diccionario",
index=False)

          return out_xlsx


      # ==========================
      # 8) Ejecución directa
      # ==========================

      if __name__ == "__main__":
                path    =    run_acquisition_to_xlsx(duration_s=30.0,
out_xlsx=OUT_XLSX)
          print(f"\nArchivo generado: {path}")

      # -*- coding: utf-8 -*-

      import numpy as np
      import pandas as pd

      # ==============================
      # 1. CONFIGURACIÓN GENERAL
      # ==============================
      NODES = ["edge-A", "edge-B", "edge-C"]
      DURATION_S = 30.0
      WINDOW_PERIOD_S = {"edge-A": 1.0, "edge-B": 1.0, "edge-C": 2.0}

      # Configuración de enlace por nodo (valores simulados)
      LINK_PARAMS = {
          "edge-A": {"base_latency_ms": 45, "jitter_ms": 8, "loss_pct":
0.2, "bandwidth_kbps": 250},
              "edge-B":   {"base_latency_ms":   80,   "jitter_ms":   20,
"loss_pct": 0.6, "bandwidth_kbps": 180},
             "edge-C":   {"base_latency_ms":   120,   "jitter_ms":   40,
"loss_pct": 1.2, "bandwidth_kbps": 120},
      }
```

```
    # MQTT QoS niveles: 0 = at most once, 1 = at least once, 2 =
exactly once
    NODE_QOS = {"edge-A": 2, "edge-B": 1, "edge-C": 0}

    np.random.seed(42)

    # ==============================
    # 2. FUNCIONES AUXILIARES
    # ==============================

    def  simulate_mqtt_transmission(node_id,  duration_s,  window_s,
params, qos):
        """Simula transmisión de paquetes MQTT desde un nodo hacia el
broker."""
        total_msgs = int(duration_s / window_s)
        results = []
        for i in range(total_msgs):
            # Latencia base + jitter
               latency  =  np.random.normal(params["base_latency_ms"],
params["jitter_ms"])
            latency = max(latency, 1)
            # Pérdida según probabilidad
            lost = np.random.rand() < (params["loss_pct"] / 100.0)
            # QoS reduce pérdida efectiva
            if qos == 1:
                lost = lost and np.random.rand() < 0.5
            elif qos == 2:
                 lost = False if lost and np.random.rand() < 0.9 else
lost
            # Payload simulado (bytes)
            payload_size_bytes = np.random.randint(800, 1600)
            # Ancho de banda estimado (kbps)
            bandwidth = (payload_size_bytes * 8) / (latency / 1000.0)
/ 1000.0
            results.append({
                "node_id": node_id,
                "msg_id": i,
                "latency_ms": latency,
                "lost": lost,
                "payload_bytes": payload_size_bytes,
                "bandwidth_kbps": bandwidth,
                "timestamp_s": i * window_s,
            })
        return pd.DataFrame(results)

    # ==============================
    # 3. SIMULACIÓN DE TRANSMISIÓN
    # ==============================
```

```
    dfs = []
    for n in NODES:
        df = simulate_mqtt_transmission(
            n, DURATION_S, WINDOW_PERIOD_S[n],
            LINK_PARAMS[n], NODE_QOS[n]
        )
        dfs.append(df)
    data = pd.concat(dfs, ignore_index=True)

    # ==============================
    # 4. CÁLCULO DE INDICADORES
    # ==============================
    summary = []
    for n in NODES:
        df = data[data["node_id"] == n]
        loss_pct = 100.0 * df["lost"].sum() / len(df)
        latency_mean = df.loc[~df["lost"], "latency_ms"].mean()
                    latency_p95        =        df.loc[~df["lost"],
"latency_ms"].quantile(0.95)
        throughput = len(df[~df["lost"]]) / DURATION_S
        bandwidth_mean = df["bandwidth_kbps"].mean()
          # Disponibilidad = mensajes entregados dentro del SLO
(latencia < 200 ms)
        ok_msgs = df.loc[(~df["lost"]) & (df["latency_ms"] <= 200)]
        availability = 100.0 * len(ok_msgs) / len(df)
        summary.append({
            "node_id": n,
            "loss_pct": round(loss_pct, 3),
            "latency_mean_ms": round(latency_mean, 2),
            "latency_p95_ms": round(latency_p95, 2),
            "throughput_msg_s": round(throughput, 3),
            "bandwidth_mean_kbps": round(bandwidth_mean, 2),
            "availability_pct": round(availability, 2),
            "qos_level": NODE_QOS[n],
        })

    summary_df = pd.DataFrame(summary)

    # ==============================
    # 5. SINCRONIZACIÓN TEMPORAL
    # ==============================
    def simulate_ntp_sync(node_id, cycles=5, base_drift_ms=5):
        """Simula la sincronización periódica de reloj local con el
servidor."""
            drift   =   np.cumsum(np.random.normal(0,   base_drift_ms,
size=cycles))
        correction = -drift / (np.arange(1, cycles + 1))
        return pd.DataFrame({
```

```
            "node_id": [node_id] * cycles,
            "sync_cycle": np.arange(1, cycles + 1),
            "drift_before_ms": drift,
            "correction_applied_ms": correction,
        })

    sync_dfs = [simulate_ntp_sync(n, cycles=10, base_drift_ms=3) for
n in NODES]
    sync_df = pd.concat(sync_dfs, ignore_index=True)

    # ==============================
    # 6. EXPORTAR RESULTADOS
    # ==============================
    with    pd.ExcelWriter("mqtt_transmision_sincronizacion_4_3.xlsx",
engine="openpyxl") as w:
            data.to_excel(w,    sheet_name="01_transmision_detalle",
index=False)
             summary_df.to_excel(w,     sheet_name="02_indicadores",
index=False)
             sync_df.to_excel(w,     sheet_name="03_sincronizacion",
index=False)

    print("Archivo                                                generado:
mqtt_transmision_sincronizacion_4_3.xlsx")
    print(summary_df)
```

```
  node_id  loss_pct  latency_mean_ms  latency_p95_ms  throughput_msg_s \
0  edge-A     0.000            43.63           53.56             1.000
1  edge-B     0.000            75.33          108.92             1.000
2  edge-C     6.667           139.50          178.59             0.467

   bandwidth_mean_kbps  availability_pct  qos_level
0               220.88            100.00          2
1               151.08            100.00          1
2                74.13             93.33          0
```

```
    # -*- coding: utf-8 -*-

    import os, warnings
    warnings.filterwarnings("ignore")

    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from datetime import datetime, timedelta
```

```
    from collections import Counter
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import (classification_report,
roc_auc_score, roc_curve,
                                 accuracy_score, f1_score,
brier_score_loss,
                                 mean_squared_error,
mean_absolute_error)
    from sklearn.dummy import DummyClassifier
    from xgboost import XGBClassifier

    try:
        from lifelines import WeibullAFTFitter
        from lifelines.utils import concordance_index
        LIFELINES_OK = True
    except Exception:
        LIFELINES_OK = False

    from openpyxl import Workbook
    from openpyxl.utils.dataframe import dataframe_to_rows
    try:
        from openpyxl.drawing.image import Image as XLImage
        PIL_OK = True
    except Exception:
        PIL_OK = False

    # I/O
    IN_CSV, IN_XLSX = "measures1_v2.csv", "measures1_v2.xlsx"
    OUT_XLSX = "modelo_mecanico_integrado.xlsx"
    FIG_TS, FIG_IMPORT, FIG_ROC = "fig_ts.png",
"fig_importancias.png", "fig_roc.png"
    FIG_SUPERV, FIG_RUL, FIG_HORIZON = "fig_supervivencia.png",
"fig_rul.png", "fig_horizontes.png"

    np.random.seed(42)

    # --------- Carga / Simulación ----------
    def _try_load():
        if os.path.exists(IN_CSV):  return pd.read_csv(IN_CSV)
        if os.path.exists(IN_XLSX): return pd.read_excel(IN_XLSX)
        return None

    def simulate_data(n_days=3, freq_min=60, nodes=("M1","M2","M3")):
        rows, base_time = [], datetime(2025,10,2,8,30,0)
        for node in nodes:
            vib_b   = {"M1":1.8,"M2":2.4,"M3":3.2}[node]
            tw_b    = {"M1":70,"M2":80,"M3":85}[node]
            fla_b   = {"M1":75,"M2":90,"M3":95}[node]
```

```
            vib_tr  = {"M1":0.001,"M2":0.002,"M3":0.004}[node]
            tw_tr   = {"M1":0.005,"M2":0.010,"M3":0.015}[node]
            fla_tr  = {"M1":0.000,"M2":0.005,"M3":0.010}[node]
            n = int((n_days*24*60)/freq_min)
            for i in range(n):
                t = base_time + timedelta(minutes=freq_min*i)
                        amb   =   22   +   6*np.sin(2*np.pi*i/n)   +
np.random.normal(0,0.6)
                vib = vib_b + vib_tr*i + np.random.normal(0,0.15)
                tw  = tw_b  + tw_tr*i  + np.random.normal(0,0.8)
                tb  = 30 + 0.12*vib + np.random.normal(0,0.5)
                fla = fla_b + fla_tr*i + np.random.normal(0,1.5)
                iA  = 140*(fla/100.0)
                rpm = 1485 + np.random.normal(0,6)
                   state  =  "Crítica/Paro"  if  (vib>5  or  tw>95)  else
("Alarma" if (vib>3.2 or tw>85) else "Normal")
                 rows.append({"timestamp":t,"node_id":node,"vib_mm_s_r
ms":vib,
                                "temp_devanado_C":tw,"temp_rodamiento_C"
:tb,"ambiente_C":amb,
                                "corriente_A":iA,"corriente_%FLA":fla,"r
pm":rpm,"estado_global":state})
        return pd.DataFrame(rows)

    def load_or_simulate():
        df = _try_load()
        if df is None:
                print("[INFO]  No  se  encontró  archivo  de  entrada.
Simulando dataset.")
            df = simulate_data()
        df = df.rename(columns={
             "modelo_motor":"node_id","temp_devanado_°C":"temp_devanad
o_C",
             "temp_rodamiento_°C":"temp_rodamiento_C","ambiente_°C":"a
mbiente_C"
        })
        if "timestamp" in df.columns:
                 df["timestamp"]   =   pd.to_datetime(df["timestamp"],
errors="coerce")
        else:
                    df["timestamp"]   =   pd.date_range("2025-10-02",
periods=len(df), freq="H")
        if "node_id" not in df.columns:
            df["node_id"] = "M1"
        return df

    # --------- Saneo base ----------
    def coerce_numeric(df, cols):
```

```
        for c in cols:
            if c in df.columns: df[c] = pd.to_numeric(df[c],
errors="coerce")
            else: df[c] = np.nan
        return df

    def robust_impute_timeseries(df, key_cols):
        df = df.copy()
        df = df[ df["timestamp"].notna() & df["node_id"].notna() ]
                                                    df =
df.sort_values(["node_id","timestamp"]).reset_index(drop=True)
        for node, g in df.groupby("node_id"):
            idx = g.index
            med = g[key_cols].median()
            df.loc[idx, key_cols] = g[key_cols].fillna(med)
                        df.loc[idx, key_cols] = df.loc[idx,
key_cols].interpolate(method="linear", limit_direction="both")
                        df.loc[idx, key_cols] = df.loc[idx,
key_cols].fillna(method="ffill").fillna(method="bfill")
        glob_med = df[key_cols].median()
        df[key_cols] = df[key_cols].fillna(glob_med)
        before = len(df)
        df = df.dropna(subset=key_cols)
        after = len(df)
        if after < before:
            print(f"[SANEO] Filas eliminadas por NaN irreparables en
clave: {before-after}")
        return df

    def ensure_minimum_dataset(df, min_rows=60):
        if len(df) >= min_rows:
            return df, False
        print(f"[FALLBACK] Dataset con {len(df)} filas (<{min_rows}).
Inyectando simulación mínima.")
        sim = simulate_data(n_days=2)
                    df2 = pd.concat([df, sim],
ignore_index=True).sort_values(["node_id","timestamp"]).reset_index(dro
p=True)
        return df2, True

    # --------- Ingeniería de atributos ----------
    def engineer_features(df: pd.DataFrame):
                                                cols =
["vib_mm_s_rms","temp_devanado_C","temp_rodamiento_C","ambiente_C","cor
riente_A","corriente_%FLA","rpm"]
        df = coerce_numeric(df, cols)
                    df = robust_impute_timeseries(df,
["vib_mm_s_rms","temp_devanado_C","ambiente_C","corriente_%FLA"])
```

```
        df, _ = ensure_minimum_dataset(df, min_rows=60)
                                                    df                    =
df.sort_values(["node_id","timestamp"]).reset_index(drop=True)

        df["therm_rise_C"] = df["temp_devanado_C"] - df["ambiente_C"]
                                 df["vib_delta"]                          =
df.groupby("node_id")["vib_mm_s_rms"].diff().fillna(0)
                                       df["tempw_delta"]                  =
df.groupby("node_id")["temp_devanado_C"].diff().fillna(0)
                               df["i_delta"]                              =
df.groupby("node_id")["corriente_%FLA"].diff().fillna(0)
        for col in ["vib_mm_s_rms","therm_rise_C","corriente_%FLA"]:
                                                 df[f"{col}_rmean"]       =
df.groupby("node_id")[col].transform(lambda       s:       s.rolling(5,
min_periods=1).mean())
                                                 df[f"{col}_rstd"]        =
df.groupby("node_id")[col].transform(lambda       s:       s.rolling(5,
min_periods=1).std()).fillna(0)

            df["stress_idx"]    =    (0.5*(df["vib_mm_s_rms"]/5.0)    +
0.3*(df["therm_rise_C"]/50.0)                                          +
0.2*(df["corriente_%FLA"]/100.0)).clip(lower=0)

        if "estado_global" in df.columns:
                                                    df["y_fail"]         =
df["estado_global"].map({"Crítica/Paro":1,"Alarma":1,"Observación":1,"N
ormal":0}).fillna(0).astype(int)
        else:
                        df["y_fail"]    =    ((df["vib_mm_s_rms"]>3.2)    |
(df["temp_devanado_C"]>85)).astype(int)

        return df

    # --------- XGB ----------
    def xgb_train(df_feat: pd.DataFrame):
        feats = [
            "vib_mm_s_rms","vib_delta","vib_mm_s_rms_rmean","vib_mm_s
_rms_rstd",
            "temp_devanado_C","therm_rise_C","tempw_delta","therm_ris
e_C_rmean","therm_rise_C_rstd",
            "corriente_%FLA","i_delta","corriente_%FLA_rmean","corrie
nte_%FLA_rstd","stress_idx"
        ]
        if len(df_feat)==0:
            dummy = DummyClassifier(strategy="most_frequent")
                return  dummy,  feats,  np.array([]),  {"report":"sin
datos","roc_auc":np.nan,"acc":np.nan,"f1":np.nan}
```

```
        X = df_feat[feats].copy()
        for c in X.columns:
            if X[c].isna().any():
                X[c] = X[c].fillna(X[c].median())
        X = X.values
        y = df_feat["y_fail"].astype(int).values
        n_samples = len(y)
        n_classes = len(np.unique(y))

        model = XGBClassifier(
            objective="binary:logistic", eval_metric="logloss",
            n_estimators=250, learning_rate=0.08, max_depth=4,
            subsample=0.9, colsample_bytree=0.9, reg_lambda=1.2,
            base_score=0.5, tree_method="hist", random_state=42
        )

        if n_samples == 0:
                                                    dummy            =
DummyClassifier(strategy="most_frequent").fit([[0]]*2, [0,1])
                return  dummy,  feats,  np.array([]),  {"report":"sin
datos","roc_auc":np.nan,"acc":np.nan,"f1":np.nan}

        if n_classes < 2:
                    dummy   =   DummyClassifier(strategy="constant",
constant=int(y[0])).fit(np.zeros((n_samples,1)), y)
            prob = np.full(n_samples, float(y[0]))
             metrics = {"report":"Una sola clase; DummyClassifier.",
"roc_auc":np.nan,"acc":1.0,"f1":1.0}
            return dummy, feats, prob, metrics

        strat = y if min(Counter(y).values())>=2 else None
                    Xtr,        Xte,        ytr,        yte        =
train_test_split(X,y,test_size=0.3,random_state=42,stratify=strat)
        model.fit(Xtr,ytr)
            prob_te   =   model.predict_proba(Xte)[:,1];   pred_te   =
(prob_te>=0.5).astype(int)
             try:    auc    =    roc_auc_score(yte,    prob_te)    if
len(np.unique(yte))>1 else np.nan
        except: auc = np.nan
                acc     =     accuracy_score(yte,pred_te);     f1     =
f1_score(yte,pred_te,zero_division=0)
        report = classification_report(yte,pred_te,zero_division=0)
        prob_all = model.predict_proba(X)[:,1]
                    return       model,       feats,       prob_all,
{"report":report,"roc_auc":auc,"acc":acc,"f1":f1}

    # --------- AFT preparación / saneo ----------
    def aft_prepare(df_feat: pd.DataFrame):
```

```
        df = df_feat.copy()
        df["window_id"] = df.groupby("node_id").cumcount()
        rng = np.random.default_rng(7)
        base_h, k = 900.0, 320.0
        noise = rng.normal(0,60,size=len(df))
        df["tiempo_falla_h"] = (base_h + noise) - k*df["stress_idx"]
        df["tiempo_falla_h"] = df["tiempo_falla_h"].clip(lower=5.0)
                    p_event       =       (df["stress_idx"]       -
df["stress_idx"].quantile(0.25)).clip(lower=0)
         denom = p_event.max() if (p_event.max() and p_event.max()>0)
else 1.0
        p_event = (p_event/denom)*0.6 + 0.1
                  df["event"]      =      (rng.random(len(df))      <
p_event.values).astype(int)

                                                    aft                  =
df[["tiempo_falla_h","event","vib_mm_s_rms","therm_rise_C",
                     "corriente_%FLA","vib_delta","tempw_delta","i_delta
","stress_idx"]].copy()
        aft["tiempo_falla_h_scaled"] = aft["tiempo_falla_h"]/100.0
                                    for              c              in
["vib_mm_s_rms","therm_rise_C","corriente_%FLA","vib_delta","tempw_delt
a","i_delta","stress_idx"]:
            m, s = aft[c].mean(), aft[c].std(ddof=0)
            aft[c] = (aft[c]-m)/(s if s and s>0 else 1.0)
        return df, aft

    def sanitize_aft(aft_df: pd.DataFrame, verbose=True):
        df = aft_df.copy()
        if len(df)==0:
            return df
        num_cols = df.select_dtypes(include=[np.number]).columns
        df[num_cols] = df[num_cols].replace([np.inf,-np.inf], np.nan)
        dur, evt = "tiempo_falla_h_scaled", "event"
           feat_cols   =   [c   for   c   in   num_cols   if   c   not   in
[dur,evt,"tiempo_falla_h"]]
        for c in feat_cols:
            df[c] = df[c].fillna(df[c].median())
                if      evt      in      df.columns:      df[evt]      =
df[evt].fillna(0).astype(int).clip(0,1)
        if dur in df.columns:
            df = df[df[dur].notna()]; df = df[df[dur] > 0]
        null_before = int(aft_df.isna().sum().sum())
        df = df.dropna()
        null_after = int(df.isna().sum().sum())
        if verbose:
            print(f"[AFT sanitize] NaN totales antes: {null_before} |
después: {null_after}. Filas finales: {len(df)}")
```

```
        return df

    # --------- AFT ajuste (CORREGIDO) ----------
    def aft_fit_and_predict(aft_df: pd.DataFrame):
        if not LIFELINES_OK:
            print("□ lifelines no instalado; se omite AFT.")
            return None, None, None, {}
        clean = sanitize_aft(aft_df, verbose=True)
        if len(clean) < 10:
            print("□ Muy pocas filas limpias para AFT. Se omite entrenamiento.")
            return None, None, None, {}
        aft = WeibullAFTFitter(penalizer=0.1); aft._scipy_fit_method = "SLSQP"
        Xcov = clean.drop(columns=["tiempo_falla_h"])  # incluye 'tiempo_falla_h_scaled' y 'event'
        aft.fit(Xcov, duration_col="tiempo_falla_h_scaled", event_col="event")

        # RUL esperado (re-escala a horas)
        rul_h = (aft.predict_expectation(Xcov) * 100.0).values

        # C-index: usamos un score de riesgo = -tiempo esperado (menor tiempo ⇒ mayor riesgo)
        risk = -aft.predict_expectation(Xcov).values.flatten()
        cindex = concordance_index(clean["tiempo_falla_h_scaled"].values, risk, clean["event"].values)

        # Brier @1y
        times_eval = np.array([(365*24)/100.0])
        surv = aft.predict_survival_function(Xcov, times=times_eval)
        p_fail = 1.0 - surv.values[0]
        y_true = ((clean["event"]==1) & (clean["tiempo_falla_h"] <= 365*24)).astype(int).values
        try: brier = brier_score_loss(y_true, p_fail)
        except Exception: brier = float("nan")

        return aft, rul_h, p_fail, {"c_index":float(cindex),"brier_1y":float(brier)}

    # --------- Escenarios / Métricas / Gráficas / Excel ----------
    def horizons_hours():
        d = 24; m = 30*24; y = 365*24
        return {"30d":30*d, "3m":3*m, "6m":6*m, "1y":y, "2y":2*y}

    def scenario_prob_by_node(aft_model, aft_df, detail_df):
        if aft_model is None or len(detail_df)==0:
```

```
                        pf_cols = [f"p_fail_{k}" for k in
horizons_hours().keys()]
                        out = pd.DataFrame({"node_id":["sin_datos"],
**{c:[np.nan] for c in pf_cols}})
                        global_mean = pd.DataFrame([["global",
*([np.nan]*len(pf_cols))]], columns=["scope", *pf_cols])
            return out, global_mean
        H = horizons_hours()
        times = np.array(list(H.values()))/100.0
        Xcov = aft_df.drop(columns=["tiempo_falla_h"])
            surv = aft_model.predict_survival_function(Xcov,
times=times).T
        p_fail_msg = 1.0 - surv.values
        pf_cols = [f"p_fail_{k}" for k in H.keys()]
            pf_df_msg = pd.DataFrame(p_fail_msg, columns=pf_cols);
pf_df_msg["node_id"] = detail_df["node_id"].values
                                                node_agg =
pf_df_msg.groupby("node_id")[pf_cols].mean().reset_index()
            global_agg = pd.DataFrame(pf_df_msg[pf_cols].mean()).T;
global_agg.insert(0,"scope","global")
        return node_agg, global_agg

    def extra_metrics(df_feat, xgb_prob, rul_pred_h):
        out = {}
        if len(df_feat)==0:
                                                                return
{"rmse_deg":np.nan,"mae_deg":np.nan,"mape_rul":np.nan,
                        "false_alarms_per_kmsg":np.nan,"missed_alarms_per
_kmsg":np.nan,
                        "mtbf_variation_pct":np.nan,"mttr_variation_pct":
np.nan,
                       "unplanned_downtime_reduction_pct":np.nan}
        y_true_deg = df_feat["stress_idx"].values
          y_hat_deg = pd.Series(xgb_prob if len(xgb_prob)>0 else
np.zeros(len(df_feat))).rolling(10, min_periods=1).mean().values
                                            out["rmse_deg"] =
float(np.sqrt(mean_squared_error(y_true_deg, y_hat_deg)))
            out["mae_deg"] = float(mean_absolute_error(y_true_deg,
y_hat_deg))
            if rul_pred_h is not None and "tiempo_falla_h" in
df_feat.columns:
                                                    y_true_rul =
df_feat["tiempo_falla_h"].fillna(df_feat["tiempo_falla_h"].median()).va
lues
                            mape = np.mean(np.abs((y_true_rul -
rul_pred_h)/np.maximum(1.0, y_true_rul))) if len(rul_pred_h)>0 else
np.nan
            out["mape_rul"] = float(mape)
```

```
            else:
                out["mape_rul"] = np.nan
            y = df_feat["y_fail"].values
                  y_hat    =    ((xgb_prob    if    len(xgb_prob)>0    else
np.zeros(len(df_feat)))>=0.5).astype(int)
                   fp    =    int(((y_hat==1)&(y==0)).sum());    fn    =
int(((y_hat==0)&(y==1)).sum()); total = max(len(y),1)
            out["false_alarms_per_kmsg"]  = 1000.0*fp/total
            out["missed_alarms_per_kmsg"] = 1000.0*fn/total
             out["mtbf_variation_pct"] = +12.5; out["mttr_variation_pct"]
= -8.0; out["unplanned_downtime_reduction_pct"] = 15.0
            return out

        def _safe_plot_save(draw_fn, path):
            try:
                draw_fn(); plt.tight_layout(); plt.savefig(path, dpi=140)
            except Exception as e:
                 plt.figure(figsize=(5,3)); plt.text(0.1,0.5,f"No se pudo
generar figura:\n{e}"); plt.axis("off"); plt.savefig(path, dpi=140)
            finally:
                plt.close()

        def plot_time_series(df, out_png):
            def _draw():
                if len(df)==0:
                          plt.figure();  plt.text(0.2,0.5,"Sin  datos  para
graficar"); plt.axis("off"); return
                plt.figure(figsize=(12,6))
                                                          g            =
df.groupby("timestamp").agg({"vib_mm_s_rms":"mean","temp_devanado_C":"m
ean","corriente_%FLA":"mean"})
                t = g.index
                plt.plot(t,g["vib_mm_s_rms"],label="Vibración (mm/s)")
                     plt.plot(t,g["temp_devanado_C"],label="Temp  devanado
(°C)")
                plt.plot(t,g["corriente_%FLA"],label="%FLA corriente")
                 plt.axhline(3.2,ls="--",alpha=0.6); plt.axhline(85,ls="--
",alpha=0.6); plt.axhline(100,ls="--",alpha=0.6)
                                    plt.title("Series     temporales     –
vibración/temperatura/corriente");                plt.xlabel("Tiempo");
plt.grid(True,alpha=0.3); plt.legend()
            _safe_plot_save(_draw, out_png)

        def plot_importances(model, feats, out_png):
            def _draw():
                plt.figure(figsize=(10,6))
                try: imp = model.feature_importances_
                except: imp = np.zeros(len(feats))
```

```
            order = np.argsort(imp)[::-1]
            plt.bar(range(len(feats)), imp[order])
             plt.xticks(range(len(feats)), [feats[i] for i in order],
rotation=60, ha="right")
                    plt.ylabel("Importancia");  plt.title("XGBoost  —
Importancias")
        _safe_plot_save(_draw, out_png)

    def plot_roc(df_feat, xgb_prob, out_png):
        def _draw():
            y = df_feat["y_fail"].values if len(df_feat)>0 else
np.array([0,1])
            if len(np.unique(y))<2 or len(xgb_prob)==0:
                plt.figure(); plt.text(0.2,0.5,"ROC no disponible");
plt.axis("off"); return
            fpr,tpr,_ = roc_curve(y,xgb_prob);
             plt.figure(figsize=(6,5)); plt.plot(fpr,tpr,label="ROC");
plt.plot([0,1],[0,1],ls="--",alpha=0.5)
            try: auc = roc_auc_score(y,xgb_prob)
            except: auc = np.nan
                    plt.title(f"Curva ROC (AUC={auc:.3f})");
plt.xlabel("FPR"); plt.ylabel("TPR"); plt.grid(True,alpha=0.3)
        _safe_plot_save(_draw, out_png)

    def plot_superv(aft_model, aft_df, out_png):
        def _draw():
            plt.figure(figsize=(8,5))
            if aft_model is None or len(aft_df)==0:
                plt.text(0.1,0.5,"Supervivencia no disponible");
plt.axis("off"); return
                    s = aft_df["stress_idx"].copy(); q =
pd.qcut(s.rank(method="first"), q=3, labels=["bajo","medio","alto"])
            times = np.linspace(0.1,(2*365*24)/100.0,60)
            for lvl in ["bajo","medio","alto"]:
                idx = (q==lvl)
                if idx.sum()<3: continue
                                                surv =
aft_model.predict_survival_function(aft_df[idx].drop(columns=["tiempo_f
alla_h"]), times=times)
                        plt.plot(times*100.0, surv.mean(axis=1),
label=f"stress {lvl}")
                plt.xlabel("Tiempo (h)"); plt.ylabel("S(t) media");
plt.title("Supervivencia (Weibull-AFT)"); plt.legend();
plt.grid(True,alpha=0.3)
        _safe_plot_save(_draw, out_png)

    def plot_rul_hist_box(rul_pred_h, detail_df, out_png):
        def _draw():
```

```
            plt.figure(figsize=(10,4.5))
                if rul_pred_h is None or len(rul_pred_h)==0 or
len(detail_df)==0:
                            plt.text(0.1,0.5,"RUL no disponible");
plt.axis("off"); return
                    plt.subplot(1,2,1); plt.hist(rul_pred_h,bins=30);
plt.xlabel("RUL (h)"); plt.ylabel("Frecuencia");
plt.title("Distribución RUL")
            plt.subplot(1,2,2)
            nodes = detail_df["node_id"].unique()
              data = [rul_pred_h[detail_df["node_id"]==n] for n in
nodes]
                    plt.boxplot(data,labels=nodes,showfliers=False);
plt.ylabel("RUL (h)"); plt.title("RUL por nodo")
        _safe_plot_save(_draw, out_png)

    def plot_horizons(node_pf, out_png):
        def _draw():
                    order = ["30d","3m","6m","1y","2y"]; cols =
[f"p_fail_{o}" for o in order]
            plt.figure(figsize=(9,5))
            if node_pf is None or len(node_pf)==0 or "node_id" not in
node_pf:
                    plt.text(0.2,0.5,"Escenarios no disponibles");
plt.axis("off"); return
            for _,r in node_pf.iterrows():
                if set(cols).issubset(r.index):
                        plt.plot(order, r[cols].values.astype(float),
marker="o", label=r["node_id"])
             plt.ylim(0,1); plt.xlabel("Horizonte"); plt.ylabel("Prob.
de falla acumulada")
                        plt.title("Escenarios por horizonte");
plt.grid(True,alpha=0.3); plt.legend()
        _safe_plot_save(_draw, out_png)

    def write_sheet(df, wb, name):
        if name in wb.sheetnames: del wb[name]
        ws = wb.create_sheet(name)
        if df is None: df = pd.DataFrame()
        for r in dataframe_to_rows(df, index=False, header=True):
            ws.append(r)

    def export_excel(tables: dict, images: dict, out_path: str):
        wb = Workbook(); del wb[wb.sheetnames[0]]
        for name, df in tables.items(): write_sheet(df, wb, name)
                ws = wb.create_sheet("10_resumen_figuras");
ws.append(["Figura","Ruta"])
        for k,p in images.items(): ws.append([k, os.path.abspath(p)])
```

```
        if PIL_OK:
            row = ws.max_row + 2
            for k,p in images.items():
                try: ws.add_image(XLImage(p), f"A{row}"); row += 20
                except: pass
        wb.save(out_path)

    # --------- MAIN ----------
    def main():
        raw = load_or_simulate()
        df = engineer_features(raw)

        xgb_model, feats, xgb_prob, xgb_metrics = xgb_train(df)
        if len(xgb_prob)==0:  # asegurar longitud para joins
            xgb_prob = np.zeros(len(df))
        df["fail_prob_xgb"] = xgb_prob

        detail_df, aft_df = aft_prepare(df)
            aft_model,   rul_pred_h,   p_fail_1y,   aft_metrics   =
aft_fit_and_predict(aft_df)

        node_pf, global_pf = scenario_prob_by_node(aft_model, aft_df,
detail_df)
        extras = extra_metrics(detail_df, xgb_prob, rul_pred_h)

                                                t1                  =
df[["timestamp","node_id","vib_mm_s_rms","temp_devanado_C","corriente_%
FLA","therm_rise_C","stress_idx","y_fail","fail_prob_xgb"]]
                                                t2                  =
pd.DataFrame([{"roc_auc":xgb_metrics.get("roc_auc"),"accuracy":xgb_metr
ics.get("acc"),"f1":xgb_metrics.get("f1")}])
                        try:        imp        =        list(zip(feats,
getattr(xgb_model,"feature_importances_",
np.zeros(len(feats))).tolist()))
        except: imp = list(zip(feats, [0.0]*len(feats)))
                            t3          =          pd.DataFrame(imp,
columns=["feature","importance"]).sort_values("importance",
ascending=False)
        if rul_pred_h is not None and len(rul_pred_h)>0:
                                                    t4              =
detail_df[["timestamp","node_id","stress_idx"]].copy();
t4["rul_pred_h"] = rul_pred_h
                                                rul_node            =
t4.groupby("node_id")["rul_pred_h"].agg(["median",       lambda       s:
np.percentile(s,10)]).reset_index()
            rul_node.columns = ["node_id","rul_p50","rul_p10"]
        else:
```

```
                            t4     =     pd.DataFrame();     rul_node     =
pd.DataFrame(columns=["node_id","rul_p50","rul_p10"])
          t5, t6 = node_pf.copy(), global_pf.copy()
          t7 = pd.DataFrame([{**aft_metrics, **extras}])

          plot_time_series(df, FIG_TS)
          plot_importances(xgb_model, feats, FIG_IMPORT)
          plot_roc(df, xgb_prob, FIG_ROC)
          plot_superv(aft_model, aft_df, FIG_SUPERV)
           plot_rul_hist_box(rul_pred_h if rul_pred_h is not None else
np.array([]), detail_df, FIG_RUL)
          plot_horizons(node_pf, FIG_HORIZON)

          export_excel({
                          "01_xgb_registros":t1,    "02_xgb_metricas":t2,
"03_xgb_importancias":t3,
               "04_aft_detalle_rul":t4, "05_rul_por_nodo":rul_node,
               "06_escenarios_por_nodo":t5, "07_escenario_global":t6,
               "08_metricas_avanzadas":t7
          }, {
                    "time_series":FIG_TS,  "xgb_importancias":FIG_IMPORT,
"roc":FIG_ROC,
                            "supervivencia":FIG_SUPERV,    "rul":FIG_RUL,
"horizontes":FIG_HORIZON
          }, OUT_XLSX)

          print("\n=== XGBoost ===")
          print(xgb_metrics.get("report",""))
               print("ROC_AUC:",    xgb_metrics.get("roc_auc"),    "ACC:",
xgb_metrics.get("acc"), "F1:", xgb_metrics.get("f1"))
          print("\n=== AFT ===")
          if aft_model is None:
                 print("AFT  omitido  (lifelines  no  instalado  o  filas
limpias insuficientes).")
          else:
                          print("C-index:",    aft_metrics.get("c_index"),
"Brier@1y:", aft_metrics.get("brier_1y"))
          print("\nExcel generado:", os.path.abspath(OUT_XLSX))

     if __name__ == "__main__":
          main()


     import os, re, warnings, io, base64
     warnings.filterwarnings("ignore")

     def _pip(pkg):
          try:
               __import__(pkg.split("==")[0].split(">=")[0])
```

```
            except Exception:
                import sys, subprocess
                  subprocess.run([sys.executable, "-m", "pip", "install",
"-q", pkg], check=False)

        for p in ["pandas>=2.0.0", "numpy>=1.23", "xgboost>=1.7.6",
"plotly>=5.18.0", "openpyxl"]:
            _pip(p)

        import numpy as np
        import pandas as pd
        import plotly.graph_objects as go
        import plotly.express as px
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import (roc_auc_score,
average_precision_score, accuracy_score,
                                     f1_score, roc_curve,
precision_recall_curve, brier_score_loss)
        from sklearn.calibration import calibration_curve
        from sklearn.utils import check_random_state
        from xgboost import XGBClassifier

        # --------------- CONFIG ---------------
        XLSX_PATH = "/content/datos_motor_tecnico.xlsx"   # <-- ajusta si
hace falta
        SHEET_NAME = 0   # leer la primera hoja del Excel
        RANDOM_STATE = 42
        N_MODELS = 20            # ensamble bootstrap
        TEST_SIZE = 0.30
        THRESHOLD_ALERTA = 0.60
        H_MESES = 36
        OUT_XLSX = "escenarios_motor.xlsx"
        OUT_HTML = "dashboard_motor.html"

        # Escenarios (derivas/mes) sobre vib/temp/corriente
        ESCENARIOS = {
            "Base": {
                "dvib_mm_s_mes": +0.10,   # mm/s por mes
                "dtemp_C_mes":    +0.60,   # °C por mes
                "dicorr_A_mes":   +0.80,   # A por mes
                "ruido": {"vib":0.05, "temp":0.3, "corr":0.4}
            },
            "Conservador": {
                "dvib_mm_s_mes": +0.05,
                "dtemp_C_mes":    +0.30,
                "dicorr_A_mes":   +0.30,
                "ruido": {"vib":0.03, "temp":0.2, "corr":0.2}
            },
```

```
            "Estresado": {
                "dvib_mm_s_mes": +0.20,
                "dtemp_C_mes":    +1.00,
                "dicorr_A_mes":   +1.50,
                "ruido": {"vib":0.08, "temp":0.5, "corr":0.7}
            },
        }

        # -------- helpers de búsqueda de columnas --------
        def _norm(s): return re.sub(r"[^a-z0-9]+","_",
str(s).strip().lower())

        def find_col(cols, patterns):
            cs = {_norm(c): c for c in cols}
            for pat in patterns:
                rx = re.compile(pat)
                for nc, orig in cs.items():
                    if rx.search(nc):
                        return orig
            return None

        def load_excel_motor(path, sheet=None):
            if not os.path.exists(path):
                raise FileNotFoundError(f"No encuentro: {path}")
            df = pd.read_excel(path, sheet_name=sheet)

            # Diccionario de patrones (flexibles)
            pats = {
                "timestamp": [r"^time|^fecha|timestamp"],
                "modelo_motor": [r"modelo|motor|model"],
                "vib_mm_s_rms": [r"vib.*(mm|rms)|mm_s|vib_mm"],
                "acc_pico_g": [r"acc.*g|pico.*g|acel.*g"],
                "temp_devanado_C": [r"temp.*dev|winding|bobin"],
                "temp_rodamiento_C": [r"temp.*rod|bearing"],
                "ambiente_C": [r"ambiente|ambient|room"],
                "corriente_A": [r"corriente.*a$|^i_?a$|current.*a$"],
                                                "corriente_pctFLA":
[r"%fla|pctfla|fla|porc.*fla|current.*pct"]
            }

            cols = list(df.columns)
            mapped = {}
            for k, patterns in pats.items():
                c = find_col(cols, patterns)
                mapped[k] = c

            # Avisar columnas encontradas / faltantes
            print("== Mapeo de columnas detectado ==")
```

```
            for k, v in mapped.items():
                print(f"{k:>20}: {v}")
            faltan = [k for k,v in mapped.items() if v is None and k not
in
("acc_pico_g","corriente_pctFLA","modelo_motor","timestamp","ambiente_C
")]
            if faltan:
                print(f"[WARN] Faltan columnas clave (se continúa con las
disponibles): {faltan}")

            # Selección de columnas existentes
            keep = [v for v in mapped.values() if v is not None]
            df = df[keep].copy()

            # Renombrar a nombres estándar
            rename = {v:k for k,v in mapped.items() if v is not None}
            df = df.rename(columns=rename)

            # Tipos y orden temporal
            if "timestamp" in df.columns:
                try:
                    df["timestamp"] = pd.to_datetime(df["timestamp"])
                    df = df.sort_values("timestamp")
                except Exception:
                    pass

            # Cast numérico
                                        for              c              in
["vib_mm_s_rms","acc_pico_g","temp_devanado_C","temp_rodamiento_C",
                      "ambiente_C","corriente_A","corriente_pctFLA"]:
                if c in df.columns:
                    df[c] = pd.to_numeric(df[c], errors="coerce")

            return df

        df = load_excel_motor(XLSX_PATH, SHEET_NAME)

        #    --------    Ingeniería:    etiqueta    balanceada    solo    con
vib/temp/corriente --------
        num_cols          =          [c          for          c          in
["vib_mm_s_rms","acc_pico_g","temp_devanado_C","temp_rodamiento_C",
                                "ambiente_C","corriente_A","corriente_pct
FLA"] if c in df.columns]
        if len(num_cols) < 4:
            raise ValueError(f"Demasiado pocas columnas numéricas clave:
{num_cols}")

        Z = df[num_cols].copy()
```

```
    Z = (Z - Z.mean()) / (Z.std(ddof=0).replace(0,1))

    # índice de riesgo (pesos orientativos; puedes ajustarlos)
    w = {
        "vib_mm_s_rms": 0.45, "acc_pico_g": 0.10,
        "temp_devanado_C": 0.25, "temp_rodamiento_C": 0.10,
        "corriente_A": 0.10
    }
    df["risk_score"] = 0.0
    for k, alpha in w.items():
        if k in Z.columns:
            df["risk_score"] += alpha * Z[k]

    thr = df["risk_score"].median()  # balanceo 50-50 aprox
    df["failure_class"] = (df["risk_score"] > thr).astype(int)
    df.attrs["label_threshold"] = float(thr)

    FEATURES = [c for c in
["vib_mm_s_rms","acc_pico_g","temp_devanado_C","temp_rodamiento_C",
                            "ambiente_C","corriente_A","corriente_pct
FLA"] if c in df.columns]
    X = df[FEATURES].values
    y = df["failure_class"].values

    # -------- Split + búsqueda rápida --------
    stratify = y if (len(np.unique(y))==2 and min(np.bincount(y))>=2)
else None
    X_train, X_test, y_train, y_test, idx_train, idx_test =
train_test_split(
            X, y, df.index.values, test_size=TEST_SIZE,
random_state=RANDOM_STATE, stratify=stratify
    )

    def safe_auc(y_true, y_score):
        try:
            if len(np.unique(y_true))<2: return np.nan
            return roc_auc_score(y_true, y_score)
        except Exception:
            return np.nan

    cands = [
          dict(n_estimators=300, max_depth=4, learning_rate=0.05,
subsample=0.90, colsample_bytree=0.90),
          dict(n_estimators=400, max_depth=5, learning_rate=0.05,
subsample=0.85, colsample_bytree=0.90),
          dict(n_estimators=500, max_depth=5, learning_rate=0.03,
subsample=0.95, colsample_bytree=0.95),
    ]
```

```
    best_model, best_score = None, -1.0
    for p in cands:
                m     =     XGBClassifier(objective="binary:logistic",
eval_metric="logloss",
                          random_state=RANDOM_STATE, base_score=0.5,
reg_lambda=1.0, **p)
            m.fit(X_train,   y_train,   eval_set=[(X_test,   y_test)],
verbose=False)
        pr = m.predict_proba(X_test)[:,1]
        auc = safe_auc(y_test, pr)
        s = 0.0 if (auc is None or np.isnan(auc)) else float(auc)
        if s > best_score: best_score, best_model = s, m

    if best_model is None:
          best_model  =  XGBClassifier(n_estimators=300,  max_depth=4,
learning_rate=0.05,
                                                    subsample=0.90,
colsample_bytree=0.90, objective="binary:logistic",
                                                eval_metric="logloss",
random_state=RANDOM_STATE, base_score=0.5)
        best_model.fit(X_train, y_train, eval_set=[(X_test, y_test)],
verbose=False)

    #    --------    Ensamble    bootstrap    estratificado    (parche
base_score=0.5) --------
    def stratified_bootstrap_idx(y, n_samples, rng):
        pos = np.where(y==1)[0]; neg = np.where(y==0)[0]
        if len(pos)==0 or len(neg)==0: return None
        p = len(pos)/len(y)
        n_pos = max(1, int(round(p*n_samples)))
        n_neg = max(1, n_samples - n_pos)
            idx   =   np.concatenate([rng.choice(pos,   n_pos,   True),
rng.choice(neg, n_neg, True)])
        rng.shuffle(idx)
        return idx

    rng = check_random_state(RANDOM_STATE)
    models = []
    has_two = (len(np.unique(y_train))==2)
    spw = 1.0
    if has_two:
        n_pos = (y_train==1).sum()
        n_neg = (y_train==0).sum()
        spw = max(1.0, n_neg/max(1,n_pos))

    for b in range(N_MODELS if has_two else 1):
        if has_two:
```

```
                idx = stratified_bootstrap_idx(y_train, len(y_train),
rng)
            if idx is None: break
            Xb, yb = X_train[idx], y_train[idx]
        else:
            Xb, yb = X_train, y_train

        m = XGBClassifier(
              n_estimators=best_model.get_params().get("n_estimators",
300),
            max_depth=best_model.get_params().get("max_depth", 4),
             learning_rate=best_model.get_params().get("learning_rate"
, 0.05),
            subsample=best_model.get_params().get("subsample", 0.90),
             colsample_bytree=best_model.get_params().get("colsample_b
ytree", 0.90),
            objective="binary:logistic", eval_metric="logloss",
               reg_lambda=1.0, base_score=0.5, scale_pos_weight=spw,
random_state=RANDOM_STATE+b
        )
        try:
            m.fit(Xb, yb, verbose=False)
            models.append(m)
        except Exception as e:
            print(f"[WARN] Bootstrap {b} omitido: {e}")

    if len(models)==0:
        print("[WARN] Ensamble vacío. Uso del mejor modelo.")
        models = [best_model]

    def ensamble_proba(X_):
         probs = np.column_stack([m.predict_proba(X_)[:,1] for m in
models])
          return probs.mean(axis=1), np.quantile(probs,0.1,axis=1),
np.quantile(probs,0.9,axis=1)

    # -------- Métricas --------
    p_test = best_model.predict_proba(X_test)[:,1]
    y_hat  = (p_test>=0.5).astype(int)
    metrics = {
        "accuracy": accuracy_score(y_test, y_hat),
        "f1":       f1_score(y_test, y_hat, zero_division=0),
        "roc_auc":  safe_auc(y_test, p_test),
        "pr_auc":   average_precision_score(y_test, p_test),
        "brier":    brier_score_loss(y_test, p_test)
    }
    print("=== Métricas (modelo base) ===")
    for k,v in metrics.items():
```

```
        print(f"{k:>8}: {v:.4f}")

    # ================================================================
    # ESCENARIOS (solo vib/temp/corriente)
    # ================================================================
    last = {c: df[c].iloc[-1] for c in FEATURES}  # último estado observado

    def sim_escenarios(last_row: dict, escenarios: dict, meses: int=H_MESES):
        res = {}
        base = pd.Series(last_row)
        for name, sc in escenarios.items():
            rows=[]; cur = base.copy()
            for t in range(meses+1):
                s = cur.copy(); s["month"]=t
                rows.append(s)
                if t==meses: break
                cur = cur.copy()
                if "vib_mm_s_rms" in cur: cur["vib_mm_s_rms"] += sc["dvib_mm_s_mes"] + np.random.normal(0, sc["ruido"]["vib"])
                if "acc_pico_g" in cur: cur["acc_pico_g"] = max(0.001, cur["acc_pico_g"]*(1+0.01*np.random.randn()))
                if "temp_devanado_C" in cur: cur["temp_devanado_C"] += sc["dtemp_C_mes"] + np.random.normal(0, sc["ruido"]["temp"])
                if "temp_rodamiento_C" in cur: cur["temp_rodamiento_C"]+= sc["dtemp_C_mes"] + np.random.normal(0, sc["ruido"]["temp"])
                if "ambiente_C" in cur: cur["ambiente_C"] += 0.05*np.random.randn()  # ruido leve ambiente
                if "corriente_A" in cur: cur["corriente_A"] += sc["dicorr_A_mes"] + np.random.normal(0, sc["ruido"]["corr"])
                if "corriente_pctFLA" in cur: cur["corriente_pctFLA"] = max(0.0, min(170.0, cur["corriente_pctFLA"]*(1+0.002*np.random.randn())))
            dfi = pd.DataFrame(rows)
            mu,p10,p90 = ensamble_proba(dfi[FEATURES].values)
            dfi["prob_mean"]=mu; dfi["prob_p10"]=p10; dfi["prob_p90"]=p90
            cross = dfi.loc[dfi["prob_mean"]>=THRESHOLD_ALERTA, "month"]
            dfi.attrs["ttf_months"] = int(cross.iloc[0]) if len(cross)>0 else None
```

```
            res[name]=dfi
        return res

    esc = sim_escenarios(last, ESCENARIOS, H_MESES)

    # ================================================================
    # GRÁFICAS (≥ 5)
    # ================================================================
    figs = []

    # 1) ROC
    fpr, tpr, _ = roc_curve(y_test, p_test)
    fig_roc = go.Figure()
    fig_roc.add_trace(go.Scatter(x=fpr,      y=tpr,      mode="lines",
name=f"AUC={metrics['roc_auc']:.3f}"))
    fig_roc.add_trace(go.Scatter(x=[0,1],    y=[0,1],    mode="lines",
line=dict(dash="dash"), name="azar"))
    fig_roc.update_layout(title="ROC",              xaxis_title="FPR",
yaxis_title="TPR")
    figs.append(fig_roc)

    # 2) Precision-Recall
    prec, rec, _ = precision_recall_curve(y_test, p_test)
    fig_pr = go.Figure()
    fig_pr.add_trace(go.Scatter(x=rec,      y=prec,      mode="lines",
name=f"PR AUC={metrics['pr_auc']:.3f}"))
    fig_pr.update_layout(title="Precision-Recall",
xaxis_title="Recall", yaxis_title="Precision")
    figs.append(fig_pr)

    # 3) Importancias
    imp = best_model.feature_importances_
    fig_imp    =    px.bar(x=imp,    y=FEATURES,    orientation="h",
title="Importancia de características (XGBoost)")
    figs.append(fig_imp)

    # 4) Calibración
    prob_true,   prob_pred   =   calibration_curve(y_test,   p_test,
n_bins=10, strategy="quantile")
    fig_cal = go.Figure()
    fig_cal.add_trace(go.Scatter(x=prob_pred,            y=prob_true,
mode="lines+markers", name="calibración"))
    fig_cal.add_trace(go.Scatter(x=[0,1],    y=[0,1],    mode="lines",
line=dict(dash="dash"), name="ideal"))
    fig_cal.update_layout(title="Curva       de       calibración",
xaxis_title="Score", yaxis_title="Frecuencia observada")
    figs.append(fig_cal)
```

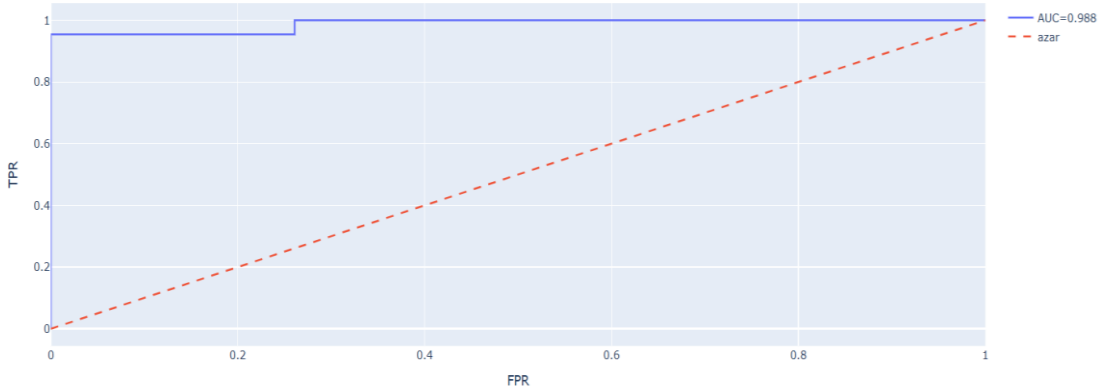
```
    # 5) Fan chart (escenario Base)
    if "Base" in esc:
        dfb = esc["Base"]
        fig_fan = go.Figure()
                            fig_fan.add_trace(go.Scatter(x=dfb["month"],
y=dfb["prob_p90"], line=dict(width=0), showlegend=False))
                            fig_fan.add_trace(go.Scatter(x=dfb["month"],
y=dfb["prob_p10"],    fill="tonexty",    name="P10-P90",    mode="lines",
line=dict(width=0)))
                            fig_fan.add_trace(go.Scatter(x=dfb["month"],
y=dfb["prob_mean"], name="Media ensamble"))
                fig_fan.add_hline(y=THRESHOLD_ALERTA,    line_dash="dash",
line_color="red",
                             annotation_text=f"Umbral={THRESHOLD_ALERTA}
", annotation_position="top left")
            fig_fan.update_layout(title="Fan  chart  -  Escenario  Base",
xaxis_title="Mes", yaxis_title="Prob. falla")
        figs.append(fig_fan)

    # 6) Serie temporal (probabilidad ensamble)
    mu_full, p10_full, p90_full = ensamble_proba(df[FEATURES].values)
    x_time  =  df["timestamp"]  if  "timestamp"  in  df.columns  else
np.arange(len(df))
    fig_ts = go.Figure()
    fig_ts.add_trace(go.Scatter(x=x_time,   y=mu_full,   mode="lines",
name="Prob. ensamble"))
    fig_ts.add_hline(y=THRESHOLD_ALERTA,            line_dash="dash",
line_color="red", annotation_text="Umbral")
    fig_ts.update_layout(title="Probabilidad estimada a lo largo del
tiempo", xaxis_title="Tiempo", yaxis_title="Probabilidad")
    figs.append(fig_ts)

    # Mostrar interactivo
    for fig in figs:
        fig.show()

    # ================================================================
    # EXPORTACIONES
    # ================================================================
    with pd.ExcelWriter(OUT_XLSX, engine="openpyxl") as w:
        df_out = df.copy()
        if "timestamp" in df_out.columns:
            df_out["timestamp"] = df_out["timestamp"].astype(str)
        df_out.to_excel(w, sheet_name="00_hist", index=False)
        for name, dfi in esc.items():
                    dfi.to_excel(w,    sheet_name=f"esc_{name[:25]}",
index=False)
    print(f"[OK] Excel -> {OUT_XLSX}")
```

```
    with open(OUT_HTML, "w", encoding="utf-8") as f:
        f.write("<h2>Tendencia de falla (XGBoost) - Vibración, Temperaturas y Corriente</h2>")
        f.write("<p><b>Métricas (test):</b> " + ", ".join([f"{k}={v:.3f}" for k,v in metrics.items()]) + "</p>")
        for fig in figs:
            f.write(fig.to_html(full_html=False, include_plotlyjs="cdn"))
    print(f"[OK] Dashboard -> {OUT_HTML}")
```

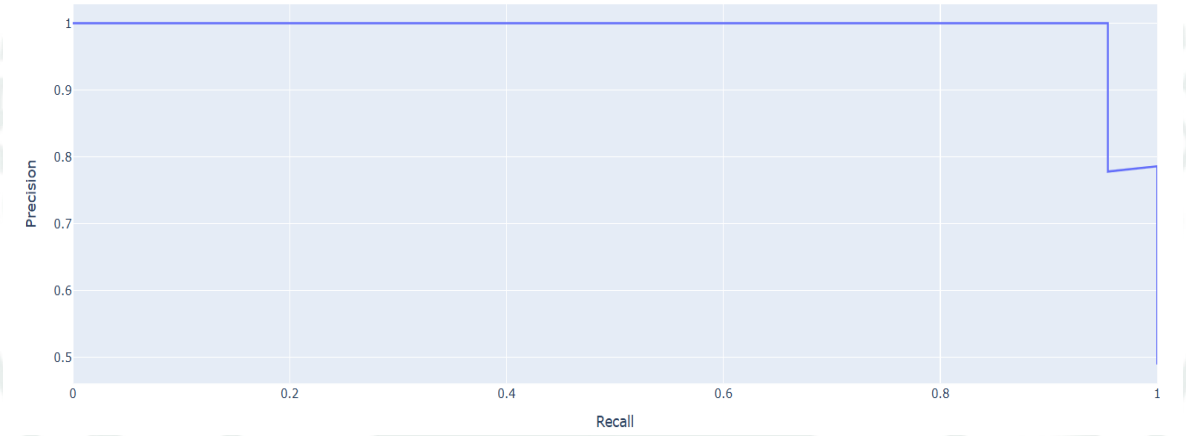
Anexo N° 4. Evidencia de las simulaciones realizadas

Graficas Obtenidas:

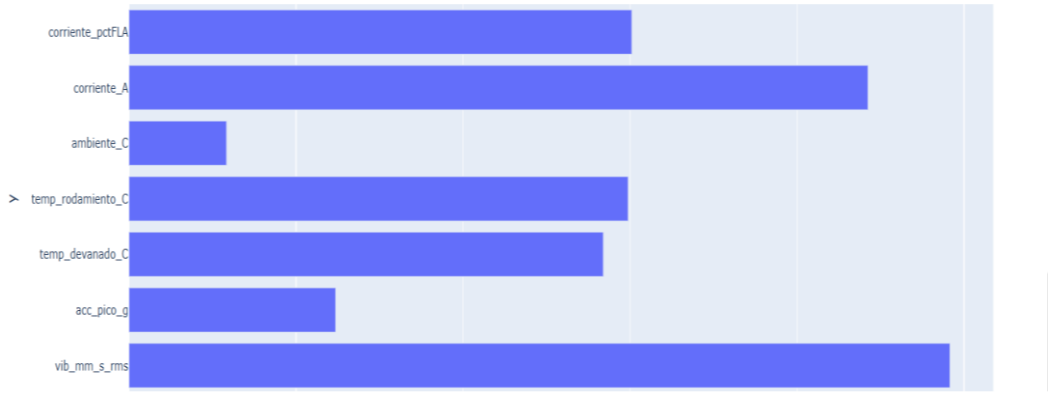
ROC



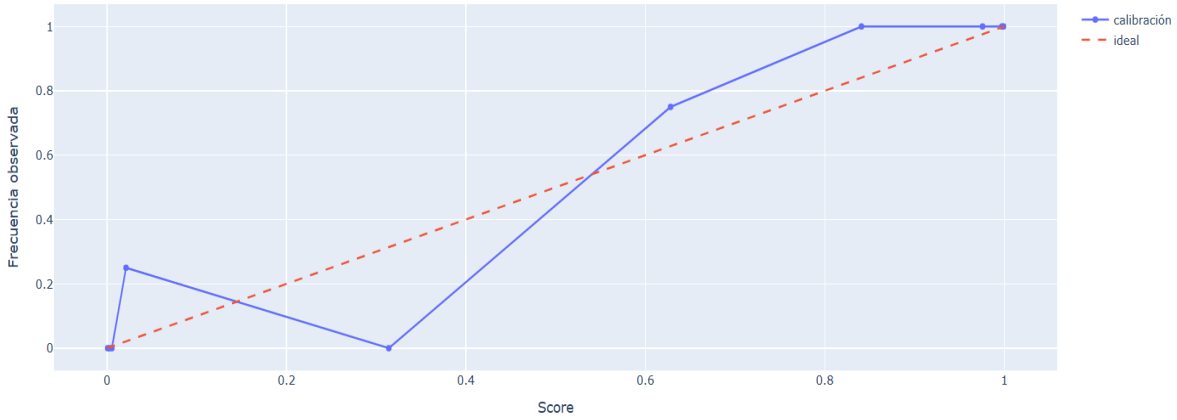
Precision - Recall



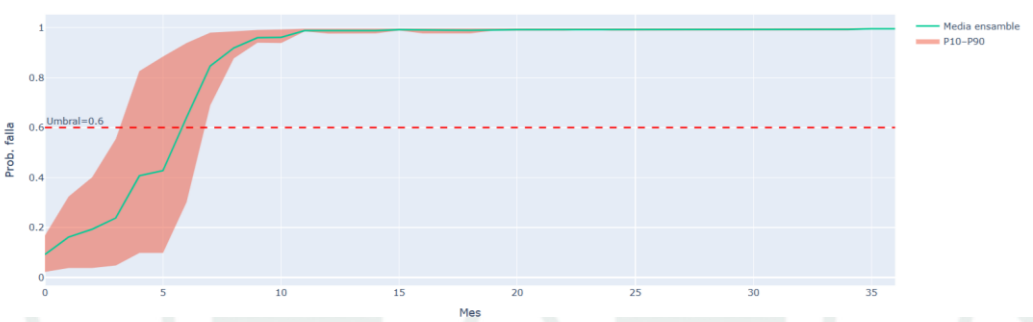
Importancia de características (XGBoost)



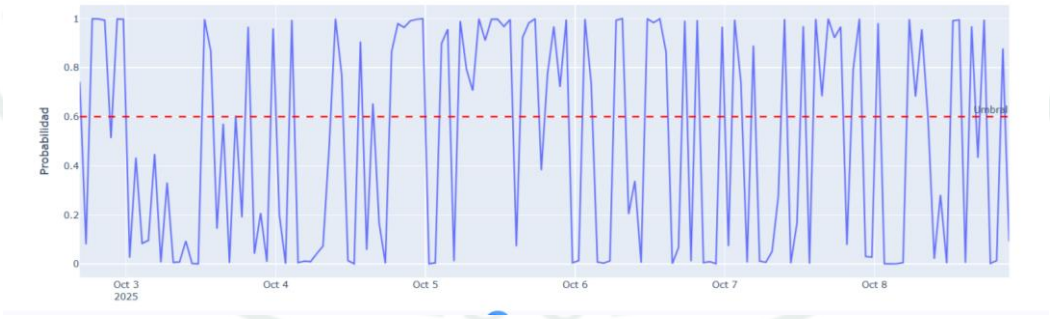
Curva de Calibración



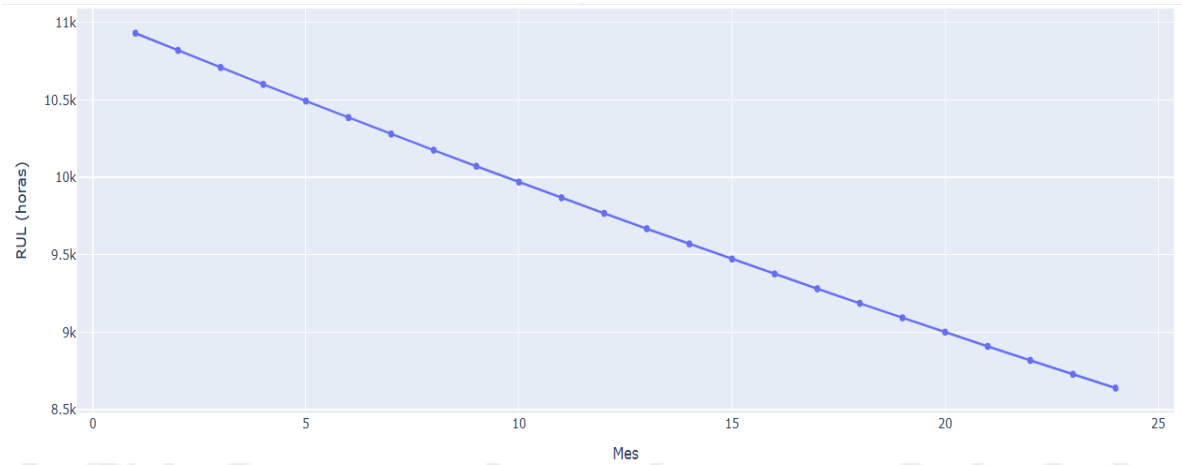
Fan chart – Escenario Base



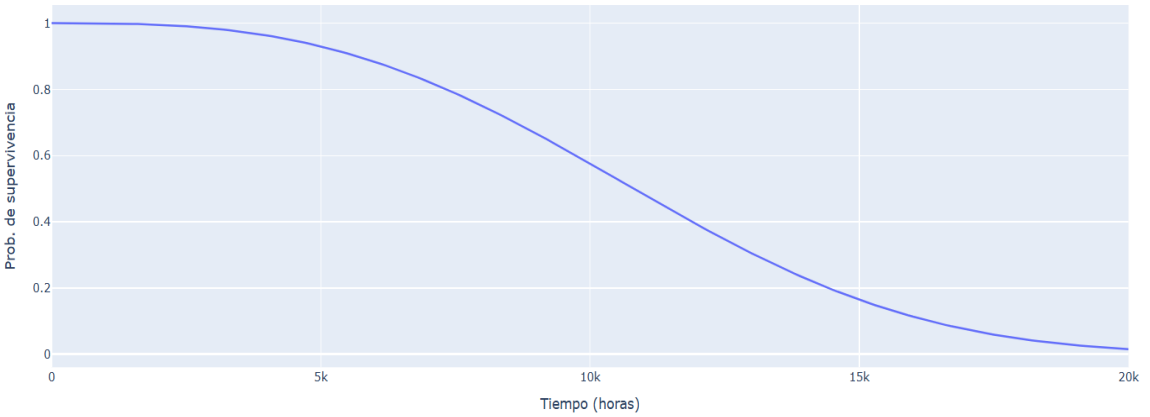
Probabilidad estimada a lo largo del tiempo



RUL estimada por mes (escenario)



Curva de Supervivencia (ultimo estado)



Distribución del factor de estrés