UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

MASTER UNIVERSITARIO EN ANALISIS DE DATOS MASIVOS (BIG DATA)

TRABAJO FIN DE MÁSTER

Arquitectura IoMT embebida para la medición del TRC con detección de alarmas mediante ML en UCI.

CLARA AIBAR ÁLVAREZ

Dirigido por

Dr. JOSE LUIS LAFUENTE CARRASCO

CURSO 2024-2025

TÍTULO: Arquitectura IoMT embebida para la medición del TRC con detección de alarmas mediante ML en UCI.

AUTOR: CLARA AIBAR ÁLVAREZ

TITULACIÓN: MASTER UNIVERSITARIO EN ANÁLISIS DE DATOS MASIVOS (BIG DATA)

DIRECTOR/ES DEL PROYECTO: Dr. JOSE LUIS LAFUENTE CARRASCO

FECHA: OCTUBRE DE 2025

RESUMEN

El TRC es un parámetro clínico de referencia para la valoración de la perfusión periférica en pacientes críticos. Aunque en los últimos años se han desarrollado dispositivos capaces de automatizar su medición, aún no existen soluciones que permitan su integración en los sistemas de monitorización hospitalarios ni en las infraestructuras de datos clínicas.

Ante esta necesidad, este trabajo presenta el diseño y validación de una arquitectura embebida e inteligente orientada a la digitalización y gestión interoperable del TRC dentro del ecosistema del IoMT. El sistema desarrollado permite la adquisición, procesamiento y transmisión estructurada de las mediciones, incorporando un modelo de ML embebido que evalúa en tiempo real la calidad de los registros y genera alertas automáticas ante mediciones no válidas. Los datos se integran en una infraestructura Big Data que posibilita su almacenamiento, análisis y visualización clínica mediante paneles interactivos.

Los resultados obtenidos confirman la viabilidad técnica de la arquitectura propuesta y su potencial para convertir el TRC en un parámetro digital continuo e interoperable, representando un avance relevante hacia una monitorización más objetiva y conectada en el manejo de pacientes críticos.

Palabras clave: tiempo de relleno capilar, IoMT, sistemas embebidos, aprendizaje automático embebido, Big Data, monitorización en cuidados críticos

ABSTRACT

CRT is a reference clinical parameter for assessing peripheral perfusion in critically ill patients. Although in recent years several devices have been developed to automate its measurement, there are still no solutions that enable its integration into hospital monitoring systems or clinical data infrastructures.

To address this need, this work presents the design and validation of an embedded and intelligent architecture aimed at the digitalization and interoperable management of CRT within the IoMT ecosystem. The developed system allows the acquisition, processing, and structured transmission of measurements, incorporating an embedded ML model that evaluates the quality of the records in real time and generates automatic alerts when measurements are not valid. The data are integrated into a Big Data infrastructure that enables their storage, analysis, and clinical visualization through interactive dashboards.

The results confirm the technical feasibility of the proposed architecture and its potential to transform CRT into a continuous and interoperable digital parameter, representing a relevant step toward more objective and connected monitoring in the management of critically ill patients.

Keywords: capillary refill time, IoMT, embedded systems, embedded machine learning, Big Data, critical care monitoring

AGRADECIMIENTOS

Quiero agradecer especialmente a mi tutor, José Luis Lafuente, por su orientación y apoyo constante durante el desarrollo de este trabajo, así como al grupo de investigación IASalud, por ofrecerme la oportunidad de formar parte de su línea y contribuir a un proyecto tan enriquecedor.

También agradezco a mi familia por su apoyo incondicional y por acompañarme en todo momento a lo largo de este camino.

TABLA RESUMEN

	DATOS
Nombre y apellidos:	CLARA AIBAR ÁLVAREZ
Título del proyecto:	Arquitectura IoMT embebida para la medición del TRC con detección de alarmas mediante ML en UCI.
Directores del proyecto:	Dr. JOSE LUIS LAFUENTE CARRASCO
El proyecto se ha realizado en colaboración de una empresa o a petición de una empresa:	NO
El proyecto ha implementado un producto: (esta entrada se puede marcar junto a la siguiente)	NO
El proyecto ha consistido en el desarrollo de una investigación o innovación: (esta entrada se puede marcar junto a la anterior)	SÍ
Objetivo general del proyecto:	Diseñar y validar una arquitectura embebida e interoperable para la digitalización y gestión del TRC que integre procesamiento local, aprendizaje automático y conectividad IoMT para la transmisión estructurada y visualización clínica de los datos en tiempo real.

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1. RESUMEN

En este capítulo se presenta una visión general del trabajo realizado, en la que se resumen el contexto y la motivación del proyecto, el planteamiento del problema, los objetivos perseguidos, los resultados obtenidos y la estructura que compone la memoria.

1.1 Contexto y justificación

La medición del tiempo de relleno capilar (TRC) es un parámetro clínico de referencia para valorar la perfusión periférica y el estado hemodinámico de los pacientes críticos. A pesar de su relevancia, su medición continúa realizándose de forma manual y subjetiva, lo que genera variabilidad y limita su integración en los sistemas de monitorización habituales en las Unidades de Cuidados Intensivos (UCI). En este contexto, el presente proyecto surge ante la necesidad de contar con una solución que permita automatizar la medición del TRC y facilitar su incorporación al entorno digital hospitalario.

1.2 Planteamiento del problema

A pesar de los avances presentes en la literatura orientados a automatizar la medición del TRC y a mejorar su precisión, siguen existiendo limitaciones importantes para su aplicación en la práctica clínica. Las soluciones actuales no generan ni transmiten las mediciones como datos estructurados ni integrables, lo que dificulta su incorporación en los sistemas de información hospitalarios y en los flujos de monitorización continua. Además, la falta de estandarización, de conectividad interoperable y de investigaciones que apliquen inteligencia artificial (IA) para su análisis, limita el desarrollo de enfoques que permitan aprovechar el TRC como parámetro clínico digital y cuantificable.

Ante esta situación, el proyecto propone una aproximación innovadora que aplica principios de procesamiento embebido e infraestructuras Big Data para convertir el TRC en un parámetro digital continuo, trazable y clínicamente útil, contribuyendo así a su integración en los entornos de monitorización y soporte a la toma de decisiones en las UCI.

1.3 Objetivos del proyecto

El proyecto tiene como objetivo principal desarrollar y validar un sistema embebido e inteligente capaz de adquirir y procesar el TRC generando datos estructurados y transmitidos en tiempo real dentro de una arquitectura del Internet de las Cosas Médicas (IoMT). Se busca integrar en una misma solución la adquisición automatizada del parámetro, la inferencia local mediante aprendizaje automático (*machine learning*, ML) y su incorporación a una infraestructura Big Data para su almacenamiento, análisis y visualización clínica.

El trabajo pretende sentar las bases para la integración del TRC en los flujos clínicos de monitorización avanzada y constituye uno de los primeros acercamientos documentados a su

digitalización e incorporación al ecosistema IoMT, representando un avance relevante para el manejo de pacientes críticos.

1.4 Resultados obtenidos

El proyecto ha dado lugar a un sistema funcional que integra la medición automatizada del TRC con capacidades de procesamiento local y conectividad IoMT. El prototipo desarrollado permite la adquisición, clasificación y transmisión estructurada de las mediciones hacia una infraestructura Big Data basada en tecnologías de streaming y almacenamiento distribuido, la cual garantiza un flujo de datos estable, trazable y con baja latencia.

En el procesamiento embebido se ha implementado un modelo de ML ligero que evalúa la calidad de la medición en tiempo real y genera alertas automáticas cuando esta no es adecuada, reduciendo así posibles falsos positivos y mejorando la fiabilidad del sistema, así como contribuyendo a mitigar la saturación de alarmas en entornos de UCI. Asimismo, se ha desplegado una interfaz de visualización clínica en Grafana que permite monitorizar en tiempo real múltiples pacientes, analizar tendencias y recibir notificaciones instantáneas durante la monitorización.

1.5 Estructura de la memoria

La memoria se organiza en distintos capítulos que reflejan el proceso completo del proyecto. Tras el resumen inicial, se presenta el contexto y el estado del arte, donde se analizan los antecedentes y las limitaciones existentes. El marco teórico recoge la documentación y el análisis de los elementos conceptuales y tecnológicos que sustentan las decisiones de diseño adoptadas en el desarrollo.
El capítulo de desarrollo detalla la implementación técnica del sistema y las evidencias que respaldan su validez, seguido de los resultados y la discusión de los hallazgos obtenidos. Finalmente, se incluyen las conclusiones, las futuras líneas de trabajo y las referencias empleadas.
Los anexos complementan la memoria principal con información técnica y materiales de apoyo que facilitan la reproducibilidad del proyecto.

Capítulo 2. ANTECEDENTES / ESTADO DEL ARTE

En este capítulo se expone el contexto en el que se enmarca este trabajo, así como la justificación de su desarrollo. En él se analizan los fundamentos clínicos y tecnológicos relacionados con el TRC, su relevancia en la valoración de la perfusión periférica y las limitaciones actuales que dificultan su uso en entornos de cuidados intensivos (CI). Además, se describe el marco general en el que surge la necesidad de digitalizar este parámetro y de integrarlo en infraestructuras IoMT y de análisis de datos clínicos, lo que sirve como punto de partida para el diseño y la validación de la solución propuesta en los capítulos posteriores.

2.1 Tiempo de relleno capilar

El TRC es un parámetro clínico ampliamente utilizado para evaluar la perfusión periférica, es decir, la eficacia con la que la sangre alcanza los capilares de la piel. Su medición es sencilla y consiste en aplicar una ligera presión sobre una zona distal del cuerpo, comúnmente el lecho ungueal de un dedo, para detener brevemente el flujo sanguíneo (ver Figura 1). Al liberar la presión, se observa y mide el tiempo que tarda en volver el color normal a la piel, lo que refleja la velocidad con la que la sangre vuelve a los capilares del dedo. (1–3)

Figura 1: Procedimiento manual de medición del TRC. (4)

En entornos críticos como las UCI, el TRC se emplea como indicador rápido y no invasivo para identificar alteraciones circulatorias, especialmente en pacientes con hipovolemia, sepsis o shock séptico, donde la perfusión cutánea suele deteriorarse de manera rápida. En relación con ello, existen diversos estudios en la literatura científica que han demostrado su valor pronóstico

en pacientes críticos. En ellos se determina que un TRC prolongado se asocia con mayor gravedad clínica y una peor evolución del pacientes, mientras que la normalización de este parámetro tras el tratamiento demuestra una mejor recuperación. Además, el TRC se trata de una medida rápida, económica, reproducible y que puede realizarse a pie de cama sin equipamiento complejo. (5, 6)

Sin embargo, pese a su sencillez y utilidad, la medición manual del TRC presenta importantes limitaciones. Su interpretación depende de factores como la presión aplicada por parte del clínico, la temperatura ambiente, la iluminación o el tono de piel del paciente, lo que genera alta variabilidad intra- e interobservador y dificulta su estandarización. Estas limitaciones reducen su fiabilidad y explican por qué, a pesar de su valor clínico probado, el TRC no se ha integrado de forma sistemática en los sistemas de monitorización digital. (7–10)

2.2 Estado del arte

La falta de objetivación del TRC ha motivado el desarrollo de tecnologías orientadas a su automatización y cuantificación, con el principal objetivo de poder realizar mediciones más precisas y reproducibles. Los avances recientes incluyen el uso de sensores ópticos, cámaras, técnicas de fotopletismografía y algoritmos de procesamiento de señal o IA, los cuales permiten registrar el proceso de reperfusión de manera cuantitativa y generar datos estructurados.

El objetivo de este apartado es revisar de forma sistemática las soluciones existentes en la literatura científica relacionadas con la medición objetiva del TRC y analizar su grado de madurez tecnológica. En concreto, se busca determinar en qué medida estas propuestas permiten avanzar hacia la digitalización e integración del TRC en sistemas clínicos conectados, considerando aspectos como la sensorización, el procesamiento embebido, la conectividad IoT/IoMT y la visualización de los datos en entornos hospitalarios. Para ello, se han establecido los siguientes criterios de comparación:

1. **Técnica de sensorización**.
2. **Procesamiento** (offline, a PC; en dispositivo; microcontrolador/TinyML).
3. **IA o algoritmos avanzados**.
4. **Conectividad e integración IoT/IoMT** (transmisión y formatos de datos).
5. **Visualización/uso clínico**.

A continuación, se revisan las principales aproximaciones tecnológicas propuestas en la literatura para la medición del TRC, clasificadas según el tipo de tecnología empleada.

Análisis de imagen / vídeo

Se pueden encontrar numerosas soluciones basadas en el análisis de imagen que cuantifican de forma objetiva el TRC analizando los cambios de color que se producen en la piel tras liberar la compresión. Para ello, se recogen secuencias de vídeo y se mide la variación de la intensidad en uno o varios canales de color, normalmente el canal verde al contar con una mayor sensibilidad. (11–13)

Este enfoque ha mostrado resultados muy prometedores. Por ejemplo, el trabajo de Bachour et al. (12) (Figura 2) logró una alta sensibilidad a los cambios microvasculares y una independencia significativa respecto al fototipo de piel, incluyendo participantes de distintos tonos cutáneos bajo condiciones controladas de iluminación y temperatura. Estos resultados confirman que el análisis por imagen puede detectar microvariaciones de perfusión que serían imperceptibles a simple vista, aportando así un gran potencial para una medición más precisa y objetiva del TRC.

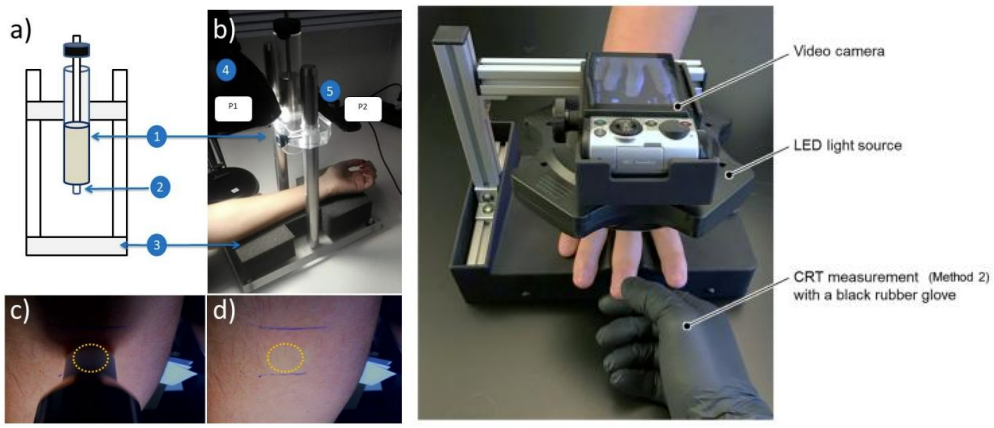


Figura 2: Ejemplos de dispositivos basados en análisis de imagen para la medición del TRC. (12)

Sin embargo, pese a sus ventajas técnicas, este tipo de sistemas aún presentan limitaciones importantes para su uso clínico real en contextos de UCI. En la mayoría de los estudios, el procesamiento se realiza de forma offline en un ordenador, lo que impide disponer del dato en tiempo real. Además, los equipos empleados suelen ser voluminosos y dependientes de condiciones controladas, lo que limita su portabilidad y su integración en entornos asistenciales como las UCI. Tampoco se describen mecanismos de transmisión de datos, ni paneles de visualización clínica, ni arquitecturas IoMT que permitan incorporar las mediciones a sistemas conectados o a registros estructurados de pacientes.

Por lo tanto, se puede afirmar de que se trata de una tecnología con un gran potencial, ya que es capaz de ofrecer medidas muy precisas y con buena consistencia cuando se controla el entorno, pero que aún no está preparada para un uso clínico. La falta de portabilidad, de procesamiento embebido y de conectividad hace que su aplicación quede restringida a entornos experimentales.

Fotopletismografía

Los sistemas basados en fotopletismografía (PPG) utilizan una fuente de luz, normalmente un LED rojo e infrarrojo, y un fotodetector colocados en contacto con la piel, de forma similar al principio de funcionamiento de un pulsioxímetro (ver Figura 3). Durante la medición del TRC, se aplica presión sobre el dedo o la zona de interés para interrumpir temporalmente el flujo sanguíneo capilar, y posteriormente se observa el cambio en la intensidad de la luz reflejada o transmitida conforme la sangre vuelve a circular. A partir de estos datos se estima de manera cuantitativa el TRC. Este método tiene la ventaja de requerir una electrónica sencilla y de bajo consumo, además de ser no invasivo y compatible con el equipamiento clínico existente. (14–18)

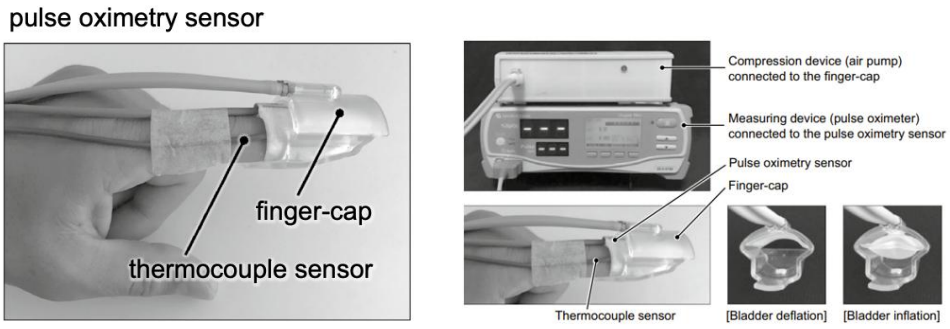


Figura 3: Ejemplo de dispositivo de medición del TRC basado en PPG. (18)

No obstante, a pesar de su utilidad y portabilidad, estas soluciones aún presentan limitaciones significativas cuando se analizan desde la perspectiva de un sistema IoMT. En primer lugar, el procesamiento de las señales suele realizarse de forma offline o en equipos externos, sin ejecución embebida que permita obtener resultados en tiempo real. En segundo lugar, la ausencia de conectividad y de mecanismos para transmitir las mediciones a un servidor o a un panel clínico impide su integración en arquitecturas conectadas. Tampoco incorporan una interfaz de visualización ni mecanismos de aseguramiento de calidad (QA) que permitan detectar artefactos o tomas incorrectas de forma automática. En términos de fiabilidad, la señal PPG es además muy sensible a movimientos, temperatura cutánea y presión de contacto, lo que puede introducir ruido y dificultar la estandarización de la medición si no se controla adecuadamente.

Los trabajos de Sheridan et al. (15) y Karlen et al. (16) introducen versiones portátiles de esta técnica, donde el procesamiento se realiza en el dispositivo o a través de una aplicación móvil (smartphone o iPod) conectada por Bluetooth. Las aplicaciones funcionan de forma local y desconectada del entorno clínico, y no presentan interoperabilidad ni transmisión de datos hacia sistemas hospitalarios o plataformas IoMT. En consecuencia, las mediciones no pueden ser supervisadas ni almacenadas de manera estructurada, lo que limita su utilidad en escenarios de monitorización continua.

Esta tecnología tiene una base tecnológica sólida y madura para la medición del TRC, con claras ventajas en términos de portabilidad, bajo coste y compatibilidad clínica, pero con un vacío funcional en lo relativo a su integración digital. Los trabajos revisados demuestran que esta tecnología mide de forma precisa parámetro clínico, pero no comunica ni procesa los datos en un entorno conectado. Por tanto, el siguiente paso lógico que todavía no ha sido abordado en la literatura científica, sería dotar a este tipo de sensores de procesamiento embebido, telemetría en tiempo real y una interfaz clínica inteligente que convierta una medición aislada en un dato útil y explotable dentro de una infraestructura IoMT.

Sensores de fibra óptica

Otra de las líneas de investigación en la medición del TRC se basa en el uso de sensores ópticos de fibra que detectan los cambios en la luz reflejada por el tejido. Estos sistemas aprovechan las propiedades de transmisión de la fibra óptica plástica (POF), la cual permite conducir la luz de forma precisa incluso en dispositivos de pequeño tamaño. Su principio de funcionamiento es muy similar al de la PPG, aunque con la diferencia de que presentan una mayor sensibilidad y pueden realizar mediciones en condiciones más controladas. (19–21)

Además, se han propuesto sistemas orientados a una monitorización continua del TRC, diseñados para entornos hospitalarios o de cuidados críticos. Sin embargo, la mayoría de los dispositivos desarrollados siguen funcionando como prototipos experimentales, conectados mediante cableado, como el dispositivo que se muestra en la Figura 4, y con el procesamiento de datos realizado en ordenador externo, lo que limita su aplicabilidad práctica. Estos sistemas no disponen de conectividad IoT, ni de procesamiento embebido que permita analizar la señal localmente. Tampoco cuentan con una plataforma de visualización que integre los datos en un contexto clínico. Esto significa que, aunque la tecnología es muy prometedora desde el punto de vista de la sensorización para el registro simultáneo de presión y la posibilidad de monitorización continua, aún no se ha llevado a cabo todavía el paso hacia su digitalización e integración operativa.

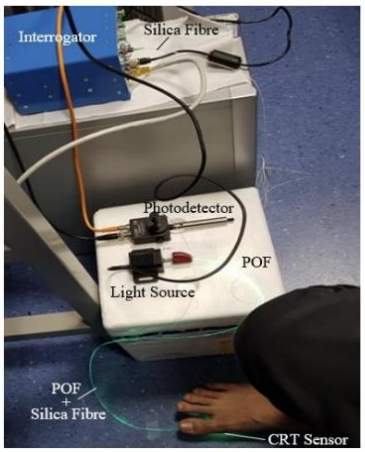


Figura 4: Sistema experimental de medición del TRC mediante sensor POF. (19)

Dispositivo portable con sensor de color

Dentro de las soluciones más recientes se encuentra el desarrollo de un dispositivo portable basado en un sensor de color, el cual representa un avance significativo hacia la estandarización de la medición del TRC en entornos clínicos reales (22, 23). Este sistema emplea un sensor RGB que detecta los cambios de tonalidad producidos en la piel durante el proceso de medición. A diferencia de otros estudios, este incorpora una pantalla integrada en el propio dispositivo que ofrece una retroalimentación visual en tiempo real al clínico durante la maniobra para indicarle tanto la presión aplicada como la duración de la compresión. En la Figura 5 se muestra la secuencia de visualización en pantalla durante el proceso de medición. De esta forma, el usuario puede ajustar su ejecución en el momento para asegurar que la medición se realiza bajo condiciones estandarizadas.

Figura 5: Dispositivo portátil para la medición automática del TRC y secuencia de visualización en pantalla durante el proceso de medición. (22)

El dispositivo realiza el cálculo del TRC de forma local sin necesidad de emplear equipos externos ni procesamiento posterior, lo que supone una mejora importante en términos de autonomía operativa y adaptabilidad al entorno de la UCI.

Sin embargo, a pesar de estos avances, este tipo de soluciones sigue presentando limitaciones importantes. Aunque el procesamiento y la guía de medición se realizan directamente en el dispositivo, no existe una infraestructura de datos asociada que permita transmitir, almacenar o analizar la información obtenida de forma centralizada. Tampoco se incluye ninguna capa de conectividad IoT/IoMT, ni mecanismos para la visualización clínica del parámetro, como dashboards, alertas o integración con sistemas hospitalarios. En consecuencia, la medición sigue siendo un proceso aislado y no presenta trazabilidad ni posibilidad de monitorización continua.

IA aplicada al TRC

Durante la revisión de la literatura, uno de los aspectos más llamativos es el escaso número de investigaciones que aplican modelos de IA al análisis del TRC. A pesar de tratarse de un parámetro con una clara utilidad clínica y de que existen dispositivos ampliamente extendidos, como los pulsioxímetros previamente mencionados, cuya tecnología está totalmente

consolidada en entornos críticos como las UCI, el uso de algoritmos de Machine Learning (ML) para su interpretación es todavía muy limitado y poco explorado. Este vacío resulta sorprendente, sobre todo teniendo en cuenta el potencial de la IA para procesar señales complejas como las del TRC, reducir la variabilidad en la medición y extraer información clínica adicional a partir de los datos obtenidos.

El único trabajo identificado que aborda de forma específica esta aplicación es el estudio de ML sobre señales de pulsioximetría, en el cual se clasifica el TRC en tres categorías: “flash”, “normal” y “prolongado” (24). En este caso, se emplearon distintos modelos de ML, como XGBoost, regresión logística y máquinas de vectores soporte (SVM), comparando sus resultados con los índices tradicionales derivados de la señal PPG. Los resultados fueron prometedores, alcanzando valores de AUC altoos y mostrando que la morfología de la onda PPG contiene información fisiológica útil para caracterizar la respuesta capilar más allá de un simple valor de tiempo.

Aun así, este trabajo se mantiene en un contexto experimental y con un procesamiento realizado en ordenador externo, sin ser integrado en dispositivos embebidos ni en entornos IoT o IoMT. Tampoco se plantean capas adicionales de visualización clínica, análisis de calidad de señal o comunicación con sistemas hospitalarios, lo que limita su aplicabilidad práctica y su escalabilidad a entornos asistenciales reales de UCI.

Pese a ello, este primer resultado abre una línea de investigación muy relevante. La IA aplicada al TRC puede desempeñar un papel clave tanto en la optimización técnica del proceso de medición (por ejemplo, para validar tomas, detectar errores o ajustar parámetros de presión y tiempo de compresión), como en la explotación clínica de los datos, ayudando a parametrizar las mediciones, detectar tendencias de deterioro o mejora en el paciente y asistir en la toma de decisiones terapéuticas. Con la evolución de los modelos ligeros y la disponibilidad de microcontroladores con capacidad de cálculo, el uso de TinyML permitiría incorporar esta inteligencia directamente en el dispositivo y posibilitar un procesamiento local y en tiempo real.

En la Tabla 1 se presenta un resumen de las principales contribuciones identificadas en la literatura, organizadas según los criterios de sensorización, procesamiento, uso de IA, conectividad y visualización clínica, los cuales han sido previamente desarrollados y son determinantes para la integración del TRC en entornos clínicos:

Tabla 1: Comparativa de trabajos sobre la medición automatizada del TRC según técnica de sensorización, procesamiento e integración tecnológica. Fuente: elaboración propia.

Año y autor	Técnica de sensorización	Procesamiento	IA/ algoritmos avanzados	Conectividad IoT / IoMT	Visualización /uso clínico
2023, Bachour et al.	Imagen estructurada (vídeo RGB con luz LED blanca y polarización circular)	Offline (PC, Matlab)	No emplea IA; procesamiento clásico basado en ajuste exponencial y criterios estadísticos automáticos de calidad	No (sin comunicación ni integración clínica; análisis en laboratorio)	No hay interfaz clínica; método experimental con análisis post-proceso
2023, Jacquet-Lagrèze et al.	Imagen estructurada (vídeo RGB); dispositivo handheld DiCART™ II	En dispositivo (edge), casi en tiempo real (interpretación ≤ 4,6 s)	No IA; algoritmo estadístico propietario	No indicada (sin transmisión ni integración clínica reportada)	Lectura en pantalla del dispositivo
2023, Rosli et al.	Sensor óptico de fibra plástica (POF)	Adquisición en Arduino; procesamiento en MATLAB en tiempo real y registro en Excel.	No; procesamiento determinista	No (sistema local sin conectividad remota)	Visualización básica en PC vía MATLAB
2023, Shinozaki et al.	sensor de color RGB	Adquisición en el dispositivo; preprocesado y cálculo en ordenador (offline)	algoritmos deterministas (Hampel, Otsu, regresión, ROC/Youden)	No (sistema local sin transmisión).	Sin interfaz clínica
2022, Shinozaki et al.	Sensor de color RGB	Integrado en microcontrolador (Arduino/C/C++)	No; umbrales y normalización deterministas	No (datos locales en microSD; sin transmisión)	Pantalla integrada con barras de fuerza/tiempo y curva de recuperación

2021, Sheridan et al.	PPG	En dispositivo (edge) vía app móvil	No (algoritmo determinista de ajuste exponencial)	Parcial (BLE a smartphone); sin integración clínica/interoperabilidad reportada	Lectura en app
2021, Ballaji et al.	PPG (fibra plástica óptica)	Offline (PC, MATLAB)	No; procesamiento determinista clásico	No (sistema de laboratorio con adquisición por cable; sin comunicación remota).	Experimental; sin interfaz clínica
2020, Hunter et al.	PPG (pulsioxímetro modificado Nihon Kohden OLV-3100)	Offline (PC); extracción de 10 features (tsfresh) y análisis en Python	Sí — ML supervisado (XGBoost, SVM-RBF y Regresión Logística)	No	Muestra CRi en pantalla; el ML es análisis post-proceso sin interfaz clínica ni uso en tiempo real
2020, Liu et al.	PPG	Offline (PC, MATLAB)	No; procesamiento determinista (ajuste exponencial clásico)	No (configuración de laboratorio con adquisición por cable; sin comunicación remota)	Prototipo experimental sin interfaz visual
2019, Oi (Yasufumi) et al.	PPG (pulsioxímetro con sensor SpO_2)	En dispositivo (Q-CRT en el equipo) + análisis estadístico offline (AUC/ROC)	No (métrica determinista Q-CRT; comparación con qSOFA/SIRS y lactato)	No indicada	Lectura local del valor Q-CRT
2016, Blaxter et al.	PPG	En microcontrolador STM32F103; post-procesamiento en GNU Octave para cálculo del CRT	Reglas deterministas, ajuste y estadística clásica (Pearson r, AUC/umbrales)	Bluetooth (RN-42) para streaming/updates; microSD para log local. Sin nube/IoMT	Software en PC
2014, Morimura et al.	PPG (pulsioxímetro modificado	Adquisición y cálculo en PC	Reglas determinísticas y estadística	Ninguna (equipo clínico autónomo; sin red).	Sin interfaz clínica

	Nihon Kohden OLV-3100)		clásica (correlación de Spearman, ROC/AUC)		
2011, Karlen et al.	PPG estándar de pulsioxímetro	Automático en app móvil	No hay ML	Dispositivo portátil offline	App en iPod
2006, Shavit et al.	Videografía digital fija (cámara USB CMOS)	Software propio	Lógica determinista de matching de color + estadística clásica (ROC, LR)	No (equipo local sin transmisión).	Pantalla del sistema muestra DCRT

2.3 Contexto y justificación

Este trabajo se enmarca en la línea de investigación del grupo IASalud, centrada en la digitalización del TRC y su integración en entornos clínicos reales. En una fase previa, se desarrolló un prototipo de un dispositivo capaz de automatizar la prueba de la medida del relleno capilar y realizar mediciones objetivas del TRC. A partir de esa base, el presente proyecto avanza hacia una etapa más ambiciosa, cuyo principal objetivo consiste en convertir esta medición en un dato digital estructurado, trazable y útil para la toma de decisiones clínicas.

Como ya se ha descrito anteriormente, el TRC es un parámetro ampliamente empleado en UCI por su valor pronóstico y su simplicidad, pero su medición manual continúa siendo subjetiva y poco estandarizada. El análisis del estado del arte muestra que, aunque existen múltiples aproximaciones tecnológicas, todas presentan limitaciones comunes. El procesamiento de datos suele realizarse offline, sin integrar una transmisión de datos en tiempo real, sin arquitecturas IoMT y sin el desarrollo de una interfaz clínica operativa. Además, las pocas investigaciones que aplican IA lo hacen en entornos de laboratorio, sin implementar una integración embebida ni conexión con sistemas hospitalarios.

Asimismo, diversos estudios han señalado que la proliferación de sistemas de monitorización en las UCI y el elevado número de alertas generadas pueden dar lugar a fenómenos de fatiga de alarmas (*alarm fatigue*), reduciendo la eficacia del sistema y aumentando el riesgo de errores por desensibilización del personal sanitario (25, 26). Es por ello por lo que resulta esencial desarrollar como parte de la solución propuesta, mecanismos que mejoren la fiabilidad de las mediciones y minimicen la generación de alertas innecesarias o imprecisas. Esta problemática constituye uno de los principales motivos que orientan el uso de ML embebido (*Tiny Machine*

Learning, TinyML) en el presente trabajo, dirigidos a evaluar la calidad de la señal y reducir la aparición de falsos positivos durante la monitorización.

En este contexto, este proyecto propone un enfoque integral orientado a superar las limitaciones detectadas en la literatura. El principal objetivo es demostrar la viabilidad técnica y funcional de un sistema que cubra el ciclo completo del dato, desde la captura hasta la visualización clínica, para poder garantizar la trazabilidad y la explotación futura de las mediciones.

Desde una perspectiva técnica, esta combinación se enmarca en el ámbito de las tecnologías Big Data, al integrar procesamiento distribuido, flujos en tiempo real, almacenamiento estructurado y analítica continua. La integración del parámetro en un sistema IoMT con múltiples dispositivos, flujos simultáneos de datos y procesamiento en streaming requiere técnicas como el tratamiento distribuido, el almacenamiento estructurado y la analítica en tiempo real.

Desde el punto de vista clínico, el proyecto contribuye a transformar un procedimiento puntual en un flujo de información continuo y objetivo, permitiendo visualizar tendencias, evaluar respuestas hemodinámicas y detectar alteraciones de forma temprana. Además, sienta las bases para que el TRC pueda integrarse plenamente en el ecosistema de datos de la UCI, lo cual permitiría que esta variable se incorpore al flujo de información clínica y contribuya realmente a la toma de decisiones en tiempo real.

Este proyecto constituye un primer acercamiento orientado a superar las limitaciones identificadas en la literatura científica, que hasta ahora han impedido la integración efectiva del TRC en la práctica clínica. A través de una arquitectura conectada, procesamiento embebido y el uso de algoritmos de IA ligera, se busca sentar las bases para su incorporación progresiva en el flujo de monitorización de las UCI.

2.4 Planteamiento del problema

Las tecnologías actuales para la medición del TRC han demostrado tener precisión en condiciones controladas, pero no se ha desarrollado su integración en un sistema operativo de uso clínico. En ellas, predominan el procesamiento offline, la ausencia de conectividad IoMT, la falta de trazabilidad y la inexistencia de visualización clínica del parámetro en tiempo real. Sin estas capas, el TRC no puede integrarse en el ecosistema de datos de la UCI ni aportar valor a la toma de decisiones clínicas.

En consecuencia, el problema que se aborda es diseñar y validar técnicamente una solución que combine: (i) cálculo embebido del TRC evaluación de la calidad de la señal (Quality Assessment, QA) y TinyML para validar mediciones y disparar alertas, (ii) telemetría fiable y un procesamiento continuo con registro histórico que facilite el análisis longitudinal de las mediciones y (iii) panel clínico orientado a flujo de trabajo en UCI. El objetivo final es transformar el TRC en un dato estructurado, continuo y accionable dentro de la infraestructura de la UCI.

Capítulo 3. OBJETIVOS

A continuación, se presentan los objetivos del proyecto, incluyendo el objetivo principal, los objetivos específicos y los aspectos que no se abordarán en el proyecto, así como los beneficios esperados del proyecto.

3.1 Objetivos generales

El presente trabajo se enmarca en la línea de investigación del grupo IASalud, orientada a la digitalización del TRC en el entorno de la UCI. En este contexto, el objetivo principal de este trabajo consiste en desarrollar y validar un sistema embebido e inteligente para la medición del TRC que integre procesamiento local, algoritmos de ML y conectividad IoMT. El sistema debe buscar constituir una solución precisa, portable y clínicamente aplicable, capaz de generar datos estructurados, emitir alertas automatizadas y facilitar la monitorización remota en entornos críticos como las UCI.

3.2 Objetivos específicos

- **Analizar el estado del arte** de las soluciones existentes para la medición del TRC, identificando sus limitaciones en precisión, reproducibilidad y capacidad de integración con entornos digitales sanitarios e IoMT.
- **Adaptar y optimizar el sistema de sensorización** óptica y de presión previamente desarrollado, garantizando la estabilidad de la medición, la sincronización de señales y la compatibilidad con una arquitectura embebida.
- **Implementar el procesamiento embebido local** en un microcontrolador ESP32-C3, permitiendo el cálculo en tiempo real del TRC y la extracción de parámetros complementarios (fuerza, iluminación, calidad de señal) sin depender de procesamiento externo.
- **Diseñar e implementar la infraestructura de comunicación IoMT** que asegure la transmisión estructurada, segura y trazable de los datos hacia una arquitectura distribuida de análisis y almacenamiento.
- **Desarrollar el flujo de ingesta y procesamiento en tiempo real** con persistencia en base de datos relacional y Data Lake, posibilitando la generación automatizada de alertas y métricas agregadas.
- **Diseñar y entrenar modelos TinyML** orientados a la validación de la calidad de las mediciones y a la detección automática de condiciones anómalas para facilitar una respuesta temprana y reduciendo la intervención manual.
- **Desarrollar una plataforma de visualización clínica**, adaptada al flujo de trabajo en UCI que permita el seguimiento de pacientes, la interpretación rápida de tendencias y la gestión categorizada de alertas.

Es importante señalar que existen aspectos que no son abordados en el presente proyecto y que son detallados a continuación:

- **Validación clínica completa:** La validación realizada se centra en el comportamiento técnico y la coherencia interna de las mediciones bajo condiciones controladas simuladas. La validación clínica con pacientes queda fuera del alcance del proyecto, ya que requiere ensayos controlados y aprobación ética específica.
- **Industrialización del dispositivo:** No se aborda la certificación médica ni la conformidad normativa del dispositivo. El desarrollo de una versión industrializable o comercial no forma parte de esta fase, cuyo foco se limita a la validación técnica y funcional como base para investigaciones posteriores.
- **Despliegue completo e integración clínica:** La integración de la arquitectura de datos se limita a un entorno de simulación mediante infraestructura local y servicios de prueba, sin conexión con sistemas hospitalarios reales. En esta fase se prioriza la validación del flujo de adquisición, almacenamiento y conectividad básica.
- **Evaluación del rendimiento del modelo TinyML:** El entrenamiento y validación de los algoritmos embebidos se realiza sobre un conjunto de datos experimental diseñado para reproducir condiciones técnicas de uso. No se pretende extrapolar sus resultados a escenarios clínicos reales ni establecer métricas de desempeño médico.

3.3 Beneficios del proyecto

El proyecto representa una evolución natural dentro de la línea de investigación del grupo IASalud, avanzando desde el desarrollo del dispositivo de medición del TRC hacia su digitalización e integración en entornos clínicos reales. Este supone el primer acercamiento a la superación de las limitaciones actuales de la medición de este parámetro en entornos clínicos, combinando de forma integrada procesamiento embebido, IA ligera y visualización clínica adaptada al flujo de trabajo en UCI.

Los principales beneficios que aporta son los siguientes:

1. **Demostración de viabilidad técnica**, al desarrollar un sistema completo capaz de calcular, transmitir y visualizar mediciones del TRC en tiempo real, validando la funcionalidad de una arquitectura IoMT aplicada a este parámetro clínico.
2. **Avance hacia la integración del TRC en la práctica clínica**, al establecer las bases técnicas y conceptuales que permiten incorporar este parámetro dentro del flujo de datos de las UCI, permitiendo su uso en el ecosistema de datos de la UCI.
3. **Aplicación de la tecnología TinyML** en el contexto del TRC, demostrando su utilidad para validar mediciones, garantizar la calidad del dato y generar alertas automáticas en tiempo real.
4. **Optimización del proceso de monitorización clínica**, al transformar una medición puntual y subjetiva en un flujo de información continuo, estructurado y analizable,

mejorando la capacidad de detección temprana de cambios hemodinámicos en pacientes críticos.

5. **Aporte metodológico y científico**, mediante una revisión del estado del arte que recoge las principales tecnologías, algoritmos y enfoques existentes, llenando un vacío de conocimiento en la literatura científica y estableciendo una base sólida para futuros desarrollos.
6. **Relevancia e innovación tecnológica**, al constituir el primer trabajo documentado que combina procesamiento embebido, IA ligera y visualización clínica orientada a la toma de decisiones en UCI.

Capítulo 4. DESARROLLO DEL PROYECTO

En este capítulo se describe la planificación, metodología y viabilidad del proyecto, junto con una síntesis de los principales resultados obtenidos. El desarrollo técnico detallado de la solución y la implementación completa se presentan en el Capítulo 6.

4.1 Planificación del proyecto

Como se ha presentado en los anteriores apartados, el presente proyecto combina componentes de hardware, arquitecturas de datos, algoritmos de ML y validación técnica en entornos clínicos simulados. Por su naturaleza multidisciplinar requiere una metodología que no solo permita gestionar el desarrollo de forma estructurada, sino que también integre criterios de validación técnica progresiva.

4.1.1 Metodología Stage-Gate

La metodología principal aplicada en este proyecto se basa en un modelo Stage-Gate adaptado a proyectos de investigación y desarrollo tecnológico experimental.

Este modelo se utiliza ampliamente en el ámbito de la innovación tecnológica y el diseño de productos complejos, ya que permite estructurar el desarrollo en fases secuenciales (stages) con entregables definidos y establecer puntos de control denominados gates en los que se evalúa el cumplimiento de los objetivos técnicos antes de avanzar a la siguiente etapa (27).

Esta estructura es especialmente útil en proyectos tecnológicos multidisciplinares que tengan componentes de distinta naturaleza, como por ejemplo hardware, software, protocolos de comunicación y análisis de datos, y en los que el avance depende de la validación progresiva y la integración de cada módulo. En este tipo de desarrollos es crucial garantizar la coherencia funcional del sistema y la trazabilidad de las decisiones técnicas. Este modelo lo facilita combinando una planificación estructurada con mecanismos de revisión continua.

En los últimos años, el modelo Stage-Gate ha evolucionado hacia versiones híbridas que incorporan elementos de metodologías ágiles, permitiendo aplicar una metodología más flexible y adaptativa (27–29). Esta combinación permite mantener una estructura de control y validación formal, mientras que introduce iteraciones dentro de cada fase que facilitan el ajuste continuo de hardware, firmware y algoritmos en función de los resultados obtenidos.

Este enfoque híbrido ha mostrado muy buenos resultados en proyectos de desarrollo tecnológico con un carácter experimental, especialmente en ámbitos como los dispositivos IoT, los sistemas complejos integrados o los entornos de prototipado rápido.

En relación con ello, el presente proyecto se ha estructurado en fases técnicas con validaciones intermedias, donde cada gate actúa como punto de revisión técnica y funcional que determina si los resultados alcanzados cumplen los criterios mínimos de calidad, precisión y viabilidad definidos al inicio.

4.1.2 Integración con prácticas ágiles y metodologías específicas por módulo

Aunque el modelo Stage-Gate define la estructura global del proyecto, dentro de cada fase se han aplicado metodologías específicas adaptadas al tipo de actividad desarrollada, lo que ha permitido llevar a cabo el proyecto de manera flexible, realizar su validación continua y garantizar la coherencia entre los distintos componentes del sistema.

Gestión ágil de tareas técnicas

El desarrollo sigue una lógica de trabajo de metodologías ágiles basada en ciclos cortos de planificación, implementación y revisión. Las tareas relacionadas con el hardware, el firmware, la comunicación IoT y la visualización se organizan en bloques iterativos, con objetivos concretos y validación funcional al cierre de cada ciclo.

Desarrollo y validación de modelos de ML

Para el diseño del modelo embebido de ML, se sigue la filosofía del proceso CRISP-DM (Cross Industry Standard Process for Data Mining), adaptada al entorno de recursos limitados del microcontrolador. El flujo de trabajo incluye la preparación y etiquetado del conjunto de datos experimental, el entrenamiento y validación del modelo en entorno Python y su optimización y conversión a formato TinyML para su despliegue en el dispositivo. Este proceso permite realizar iteraciones rápidas y reproducibles, así como ajustar el modelo en función del resultado obtenido.

Documentación y revisión técnica con el grupo de investigación

Cada fase del desarrollo genera documentación técnica e informes intermedios que se revisan de forma periódica con el equipo del grupo IASalud, compuesto por profesionales técnicos y clínicos. Estas revisiones permiten verificar la correcta funcionalidad de cada elemento, así como asegurar la alineación con los requisitos de uso clínico y mantener la trazabilidad de las decisiones de diseño tomadas. Además, la documentación generada es una base imprescindible para la transferencia de conocimiento y la continuidad de la línea de investigación en desarrollos posteriores.

4.1.3 Cronograma

La planificación del proyecto sigue la estructura de la metodología híbrida Stage-Gate, que, como se ha mencionado anteriormente, organiza el desarrollo en fases técnicas consecutivas y establece hitos intermedios de validación.

Cada una de estas fases se vincula con los principales módulos del sistema: diseño de hardware, comunicación IoT, procesamiento y almacenamiento de datos, desarrollo del modelo de ML y visualización. Para ello se incorporan iteraciones internas que permiten ajustar las funcionalidades y comprobar los resultados antes de avanzar al siguiente bloque. Las mejoras y ajustes se integran de forma gradual al completarse las validaciones técnicas de cada componente.

La Figura 6 muestra el cronograma general del proyecto, en el que se recogen las principales actividades, su duración aproximada y su correspondencia con las fases metodológicas definidas.

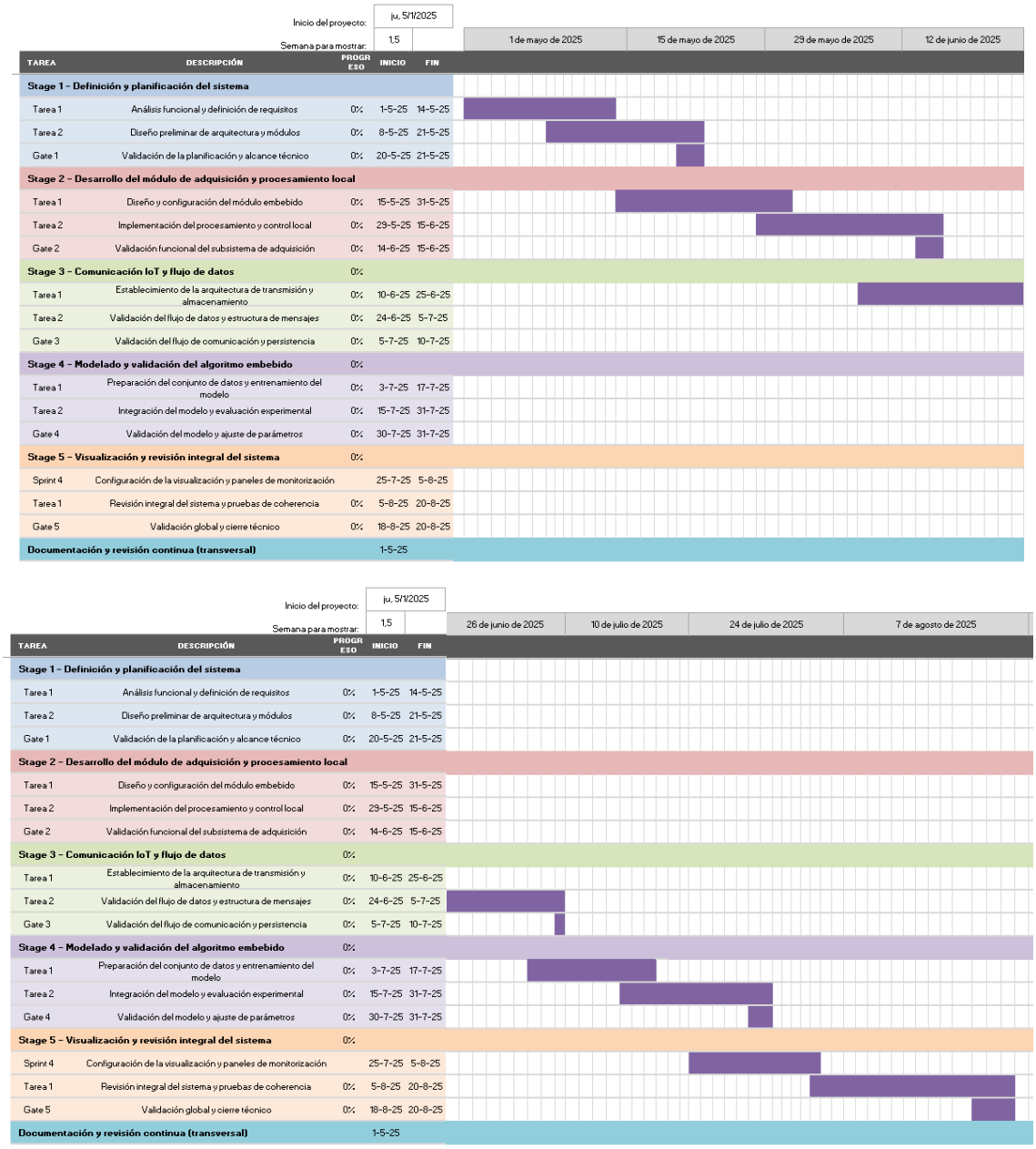


Figura 6: Diagrama de Gantt del cronograma de desarrollo del proyecto. Fuente: elaboración propia.

4.2 Descripción de la solución, metodologías y herramientas empleadas

El proyecto desarrolla una solución embebida e interconectada para la medición automatizada del TRC, diseñada para proporcionar mediciones objetivas y estructuradas con potencial de integración en entornos clínicos de CI.

El sistema implementado combina procesamiento embebido, conectividad IoT y análisis de datos para ofrecer mediciones del TRC validadas en tiempo real mediante un modelo de Tiny ML, transmitidas de forma estructurada y presentadas en una interfaz clínica que permite monitorizar su evolución de manera continua y fiable.

La arquitectura propuesta se compone de cuatro bloques funcionales principales:

1. **Adquisición y procesamiento embebido**, donde un microcontrolador de bajo consumo mide las señales ópticas y de presión y calcula el valor del TRC en tiempo real.
2. **Transmisión y almacenamiento de datos**, mediante un flujo de comunicación IoT basado en mensajería MQTT, que asegura el envío estructurado de las mediciones hacia un entorno de ingesta y almacenamiento local, así como con una base de datos adicional para el registro histórico.
3. **Análisis y visualización**, que incluye la aplicación de un **modelo de TinyML** para la detección de mediciones anómala
4. Una **plataforma de monitorización desarrollada en Grafana** para la representación en tiempo real y la revisión de alertas y tendencias.

Desde el punto de vista metodológico, el desarrollo ha seguido la metodología Stage-Gate adaptada, que ha estructurado el proyecto en fases técnicas con puntos de validación intermedia. Cada etapa ha incluido iteraciones internas para refinar el diseño y validar los resultados parciales, como, por ejemplo, la verificación del firmware, la estabilidad del flujo de datos o la precisión del modelo embebido, manteniendo una revisión técnica continua con el grupo de investigación IASalud.

Las herramientas empleadas abarcan desde entornos de desarrollo embebido hasta plataformas de análisis y visualización de datos. Para la parte de firmware se ha utilizado Arduino IDE junto con librerías específicas para el control del sensor y la comunicación MQTT. El entrenamiento y validación del modelo de ML se ha realizado en Python, empleando librerías como scikit-learn, NumPy y Pandas, con una posterior conversión a formato TensorFlow Lite Micro para su despliegue en el microcontrolador. Para el procesamiento y gestión de datos se ha empleado Apache Spark Structured Streaming y un entorno de base de datos PostgreSQL que ha sido configurado como repositorio temporal para el análisis y la visualización. Además, se ha implementado un sistema de almacenamiento histórico basado en archivos estructurados (formato Parquet), destinado a conservar los datos procesados y facilitar la trazabilidad de las mediciones. Finalmente, la supervisión de las mediciones y alertas se ha implementado

mediante Grafana, con paneles diseñados para adaptarse al flujo de trabajo del personal clínico en UCI.

En relación con las dimensiones del Big Data, el sistema aborda sus principales ejes:

- **Variedad**, ya que se integran datos de naturaleza heterogénea (ópticos, de presión y derivados del procesamiento).
- **Veracidad**, mediante el control de calidad de la señal y la validación automática de mediciones a través del modelo TinyML.
- **Velocidad** con el procesamiento embebido y la transmisión en tiempo real.
- **Volumen**, a través de la arquitectura diseñada para escalar hacia entornos clínicos donde se generen registros continuos.

Este conjunto de tecnologías y metodologías da lugar a una solución sólida que integra el procesamiento embebido, la conectividad IoT y el ML para ofrecer una arquitectura flexible y orientada al contexto clínico.

4.3 Presupuesto

A continuación se presenta en la Tabla 2 una estimación del coste económico del trabajo. Aunque este proyecto no ha requerido una financiación externa, se valoran los recursos materiales, herramientas y tiempo invertido por la autora para reflejar el esfuerzo real asociado al desarrollo de la solución.

Tabla 2: Desglose de costes estimados del proyecto según recursos materiales, técnicos y de software. Fuente: elaboración propia.

<table>
<tr><th>Tipo de coste</th><th>Valor</th><th colspan="3">Comentarios</th></tr>
<tr><td>Horas de trabajo en el proyecto</td><td>200 horas</td><td colspan="3">-</td></tr>
<tr><td>Equipo técnico utilizado</td><td>820 €</td><td colspan="3">Ordenador de sobremesa (AMD Ryzen 7 3800X · 16 GB RAM · SSD 512 GB · GPU GTX 1650)</td></tr>
<tr><td rowspan="3">Software utilizado</td><td rowspan="3">69 €</td><td>Herramienta</td><td>Tipo de uso</td><td>Coste comercial</td></tr>
<tr><td>Arduino IDE</td><td>Desarrollo firmware</td><td>0 €</td></tr>
<tr><td>Microsoft Office 365</td><td>Redacción y documentación</td><td>69 €</td></tr>
</table>

		Python + librerías (NumPy, scikit-learn, Pandas, TensorFlow Lite)	ML y análisis de datos	0 €
		Apache Spark Structured Streaming	Procesamiento de datos	0 €
		Grafana	Visualización y dashboards	0 €
		PostgreSQL	Base de datos	0 €
		Zotero	Gestión bibliográfica	0 €
Estudios e informes	0 €	Uso de recursos gratuitos para la obtención de estudios e informes científicos, como Google Scholar, PubMed, IEEE Xplore, ScienceDirect, etc.		
Materiales empleados	86,5 €	• Kit básico de Arduino: set de cables, resistencias, LEDs, protoboard, etc.: 39.09 € • Placa de desarrollo XIAO Esp32: 30.88 € • Sensor de color TCS34725: 8.59 € • Sensor FSR UX 402: 7.91 €		

4.4 Viabilidad

El proyecto se desarrolla dentro de la línea de investigación del grupo IASalud orientada al diseño de soluciones tecnológicas para la monitorización avanzada del paciente crítico. Esta línea ha dado lugar a trabajos y publicaciones centradas en la automatización de la medición del TRC. El presente trabajo representa un avance sobre las fases previas de la investigación, en las que se diseñó y validó un prototipo funcional centrado en la sensorización y el cálculo básico del TRC. A partir de esa base, el proyecto actual incorpora procesamiento embebido, algoritmos de TinyML y un sistema de visualización adaptado al entorno clínico para ampliar el alcance técnico y funcional de la solución original.

Desde el punto de vista **técnico**, el sistema demuestra una alta viabilidad para su uso en entornos controlados o de investigación gracias a su diseño modular y a la integración de componentes de bajo consumo y bajo coste. Al emplear procesamiento local, comunicación IoT y un modelo embebido de clasificación de la calidad de señal, se permite que el dispositivo opere de manera autónoma y eficiente, además de que se minimizan dependencias de infraestructura externa. Además, la arquitectura de datos desarrollada basada en Spark y PostgreSQL ofrece

una capacidad de escalado y compatibilidad con futuras integraciones en infraestructuras hospitalarias o plataformas IoMT.

En términos **económicos**, el desarrollo se ha realizado con recursos accesibles, utilizando componentes comerciales de bajo coste y software de código abierto. Esta decisión garantiza la reproducibilidad y continuidad del proyecto sin costes adicionales de licencias, lo cual facilita su posible transferencia o replicación en otros contextos académicos o clínicos. El presupuesto total estimado de 975,5 € refuerza la viabilidad económica del prototipo en estas fases tempranas de investigación.

En cuanto a la viabilidad **científica y de continuidad**, el proyecto consolida una base sólida para trabajos posteriores dentro de la misma línea de investigación. El sistema desarrollado permite recopilar datos estructurados y etiquetados, lo que abre la posibilidad de ampliar los conjuntos de entrenamiento y abordar en el futuro estudios con mayor valor clínico o con validación en pacientes reales. Los resultados obtenidos se han compartido en el marco del grupo de investigación y han contribuido a la difusión de esta línea tecnológica en congresos y publicaciones científicas recientes, lo que demuestra su relevancia y potencial de impacto.

Por último, desde el punto de vista de **sostenibilidad** y aplicación, el uso de algoritmos de TinyML y procesamiento local convierte la solución en una alternativa portátil, eficiente y viable en entornos con recursos limitados, como UCIs o contextos de emergencia. Además, el proyecto se alinea con varios de los Objetivos de Desarrollo Sostenible (ODS) (30), en particular con el ODS 3 (Salud y bienestar) y el ODS 9 (Industria, innovación e infraestructura), al promover el uso de tecnologías accesibles e interoperables para mejorar la monitorización de pacientes críticos. El empleo de componentes de bajo coste y software de código abierto contribuye a la **sostenibilidad económica** y a la **equidad en el acceso** a soluciones tecnológicas avanzadas en entornos sanitarios con recursos limitados. Asimismo, la reducción de la carga asistencial derivada de la automatización del proceso favorece la **sostenibilidad social**, optimizando el tiempo del personal clínico y mejorando la calidad de la atención.

4.5 Resultados del proyecto

En este capítulo se recogen los resultados del sistema IoMT desarrollado mostrando: (i) la arquitectura final y su operación en tiempo real, (ii) la integración IoT y el procesamiento continuo con Spark y PostgreSQL, (iii) el desempeño del modelo TinyML embebido para control de calidad y alertas y (iv) la visualización clínica en Grafana. Estos resultados confirman que el sistema funciona de manera estable, con un procesamiento rápido y una visualización clara y útil para el seguimiento del TRC en la UCI.

El detalle del diseño, el desarrollo técnico y la implementación de cada uno de estos componentes se encuentra descrito en el **Capítulo 6**, donde se documentan tanto las decisiones técnicas concretas adoptadas durante la fase de desarrollo como su respaldo en la literatura y en los fundamentos expuestos en el apartado 5.2 Fundamentos de la arquitectura IoMT y del

procesamiento embebido del Capítulo 5 Marco Teórico. Este capítulo reúne, además, las evidencias experimentales y los resultados que validan la solución implementada.

4.5.1 Descripción general del sistema final

El sistema desarrollado integra los distintos componentes de hardware y software en una arquitectura modular que permite la adquisición, el procesamiento y la visualización en tiempo real de las mediciones del TRC para su uso en contextos de UCI. La Figura 7 muestra la arquitectura completa implementada, en la que se observan las diferentes capas que componen el flujo de datos desde el dispositivo embebido hasta la interfaz de visualización clínica.

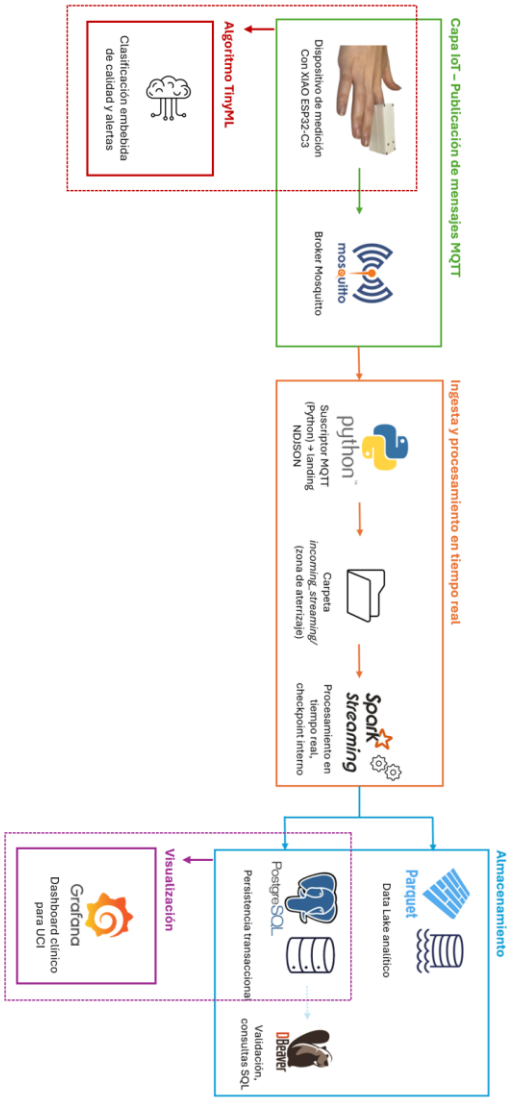


Figura 7: Arquitectura general del sistema desarrollado para la medición automatizada del TRC. Fuente: elaboración propia.

El dispositivo de medición está basado en un microcontrolador *XIAO ESP32-C3, el cual* recoge y analiza los datos ópticos y de presión para calcular el TRC, además de ejecutar localmente un modelo TinyML encargado de clasificar la calidad de la señal y generar alertas automáticas según las condiciones de medida. Las mediciones procesadas se publican mediante el protocolo MQTT a través del *broker* Mosquitto utilizando mensajes en formato NDJSON que contienen las variables estructuradas del evento (timestamp, valor de TRC, índice de calidad, etiqueta del modelo y dispositivo).

La capa de ingesta y procesamiento en tiempo real se compone de un suscriptor Python que actúa como puente entre el *broker* y el entorno de análisis. También se inluye un flujo gestionado con Spark Structured Streaming, el cual es el encargado de procesar los mensajes, mantener el estado del *streaming* mediante *checkpoints internos* y almacenar los resultados en dos destinos paralelos. Por un lado, una base de datos PostgreSQL utilizada para la consulta operativa y la visualización, y por otro, un repositorio en formato Parquet definido para el almacenamiento histórico y la explotación analítica en fases futuras de la línea de investigación.

Por último, los datos almacenados en PostgreSQL se conectan con la plataforma Grafana en tiempo real, en la que se presentan de forma estructurada a través de un *dashboard* clínico compuesto por distintos paneles diseñados para adaptarse al flujo de trabajo en las UCI. Además de su diseño funcional, el panel ofrece información analizada y clínicamente relevante, como series temporales del TRC, tendencias de evolución de cada paciente y distribución de alertas categorizadas, lo que permite una interpretación rápida del estado hemodinámico y una herramienta valiosa para la gestión del paciente crítico.

4.5.2 Integración IoT y flujo de datos

La arquitectura IoT desarrollada asegura la comunicación continua y fiable entre todos los elementos del sistema, desde el dispositivo embebido hasta la capa de almacenamiento y visualización. Su función principal es garantizar que cada medición generada en el entorno local llegue de forma íntegra y trazable al entorno analítico.

Se lleva a cabo la validación del flujo completo de transmisión de datos entre el dispositivo embebido, el broker MQTT, el sistema de procesamiento y la base de datos. La Figura 8 muestra el esquema del pipeline, donde el dispositivo publica los datos mediante MQTT, el *broker* Mosquitto gestiona la comunicación y Spark Structured Streaming actúa como consumidor y procesador en tiempo real antes de su almacenamiento en PostgreSQL.

Figura 8: Flujo de datos e integración de componentes del sistema para la adquisición, procesamiento y almacenamiento de mediciones del TRC. Fuente: elaboración propia.

Durante las pruebas, el flujo implementado mostró una buena estabilidad y baja latencia, sin perder mensajes y con una correcta estructuración de los datos en formato NDJSON. La persistencia y la trazabilidad de las mediciones se verificaron mediante consultas SQL y comprobaciones directas sobre los registros almacenados, con los que se confirmó la correspondencia entre los eventos publicados y los datos procesados. Además, el sistema mantiene una respuesta bidireccional inmediata y es capaz de reconectarse automáticamente tras cualquier interrupción de red, garantizando la continuidad del servicio y la integridad del flujo de datos.

Estos resultados confirman la fiabilidad y robustez del sistema de comunicación, lo cual garantiza un intercambio de información adecuado entre el entorno embebido y la capa de almacenamiento y visualización.

4.5.3 Modelado TinyML y resultados de validación

Dentro del sistema, el modelo TinyML embebido cumple una función clave, ya que garantiza la calidad y la coherencia de las mediciones antes de su envío al entorno de análisis. Este componente dota al dispositivo de autonomía e inteligencia, y permite enriquecer los datos para la generación de alarmas sin depender de un procesamiento externo.

El modelo TinyML embebido en el microcontrolador se entrena a partir de un conjunto de mediciones parametrizadas según variables técnicas del dispositivo, como el TRC y el índice de calidad de señal, entre otras. La definición y extracción de estos parámetros, necesarios para representar de forma cuantitativa la calidad y estabilidad de cada medición, constituyen un paso esencial, ya que permiten estructurar un conjunto de datos propio y reproducible para el entrenamiento del modelo.

El modelo final supervisado clasifica cada medición en tres categorías: OK, BORDERLINE o REPEAT, en función de su grado de fiabilidad y las condiciones técnicas detectadas, aprendidas a partir de los patrones observados en los datos. Durante la validación cruzada, el modelo presenta una precisión media del 87 %, mostrando una buena capacidad de generalización. La integración del modelo en el propio dispositivo permite que cada medición sea evaluada en

tiempo real y que se genere automáticamente su etiqueta de calidad para transmitirla posteriormente junto con los datos mediante MQTT.

Este enfoque resulta especialmente relevante en el contexto de las UCI donde es fundamental disponer de mediciones fiables y consistentes. La detección automática de mediciones anómalas permite evitar que datos inestables o de baja calidad se interpreten como válidos, al mismo tiempo que reduce la aparición de falsos positivos, manteniendo un equilibrio entre sensibilidad y especificidad que resulta esencial para la aplicación clínica. De este modo, el modelo se trata de una herramienta imprescindible para garantizar la seguridad y utilidad práctica del sistema en escenarios reales de monitorización crítica.

4.5.4 Visualización clínica

La capa de visualización constituye el punto de contacto entre la tecnología y el usuario clínico. A través de ella, las mediciones del TRC se transforman en información comprensible y de valor práctico para la supervisión del paciente crítico.

Esta visualización de resultados se implementó en Grafana está conectada directamente a la base de datos PostgreSQL. Se diseñó un dashboard específico para el personal clínico de UCI, con el objetivo de ofrecer una visión rápida y contextualizada del estado de los pacientes, las tendencias del TRC y las alertas activas.

La Figura 9 muestra el *dashboard* principal, compuesto por distintos paneles complementarios que reflejan las distintas etapas del proceso de monitorización:

- Vista global de estabilidad de la unidad
- Resumen de pacientes activos con su valor de TRC, tendencia y etiqueta de calidad más reciente
- Panel de estado instantáneo con codificación cromática (verde: TRC normal, rojo: TRC prolongado)
- Gráfica de tendencia temporal para detectar variaciones progresivas
- Panel de alertas recientes que agrupa los eventos recientes según su tipo y severidad, y permite realizar un seguimiento rápido de los eventos críticos

La integración del panel con la arquitectura IoT permite que las alertas generadas localmente por el dispositivo se reflejen de forma inmediata en la interfaz de visualización, facilitando la monitorización continua y la trazabilidad completa de cada medición. El diseño prioriza la claridad, la inmediatez y la categorización visual, de modo que la información más relevante se interprete de un vistazo, lo cual favorece a una toma de decisiones rápida y fundamentada en entornos de alta demanda asistencial como las UCI.

Figura 9: Dashboard clínico desarrollado en Grafana para la visualización y monitorización en tiempo real del TRC, compuesto por paneles de estado, tendencias y alertas automáticas. Fuente: elaboración propia.

Capítulo 5. MARCO TEÓRICO

En este capítulo se tratan los fundamentos técnicos y conceptuales que sustentan el desarrollo del sistema propuesto. En primer lugar, se describe el dispositivo de medición y los principales componentes de sensorización empleados. A continuación, se presentan los fundamentos de la arquitectura IoMT y del procesamiento embebido que son explorados con el objetivo de determinar, durante la fase de desarrollo, las tecnologías más adecuadas y las decisiones de diseño adoptadas en base a la documentación técnica y la literatura especializada. Finalmente, se analizan las ventajas y los desafíos asociados a la implementación de este tipo de soluciones en entornos clínicos.

5.1 Dispositivo de medición

Tal y como se ha indicado en apartados anteriores, el dispositivo utilizado en este trabajo procede de una etapa previa dentro de la misma línea de investigación del grupo IASalud y constituye la base experimental sobre la que se apoya la adquisición de datos del TRC (véase Figura 10). (31)

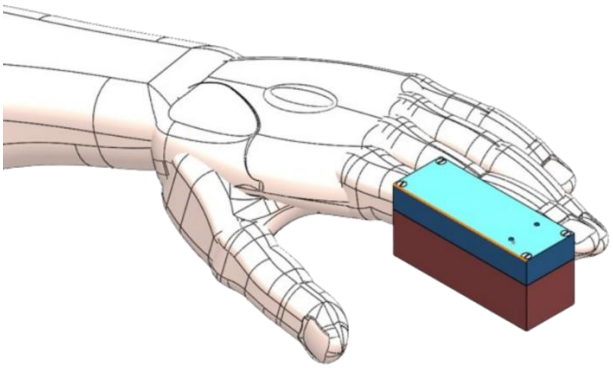


Figura 10: Modelo 3D del dispositivo de medición del TRC utilizado como base experimental. (31)

Este sistema integra dos tipos de sensorización para el cálculo del TRC. Por un lado una sensorización óptica, que registra los cambios de color producidos durante la reperfusión de la uña y otra de presión, que permite medir la fuerza aplicada durante el proceso de compresión, así como detectar el inicio y el final de esta para determinar el momento en que comienza el TRC. Esta combinación permite que las mediciones sean estandarizadas y reproducibles. En la Figura 11 se muestra el esquema general de ambos sistemas de sensorización y las curvas de datos obtenidas a partir de ellos.

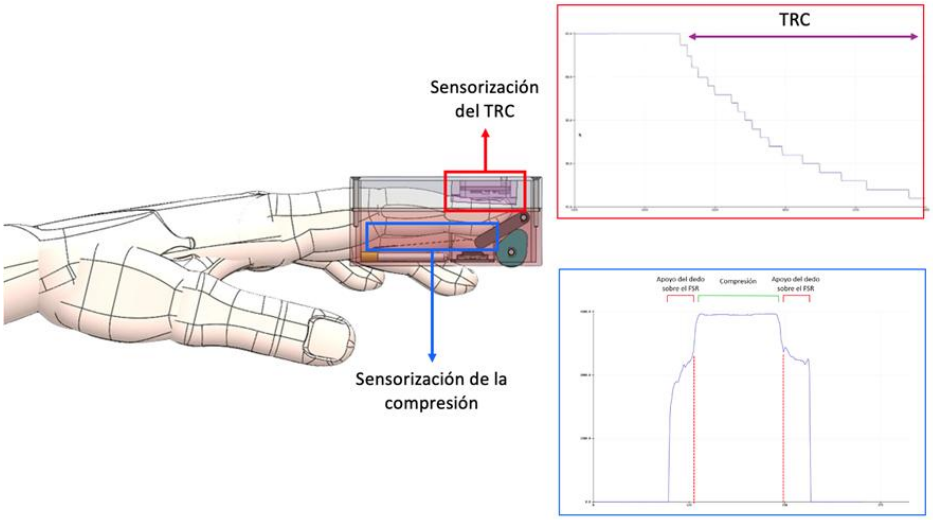


Figura 11: Esquema funcional del dispositivo de medición del TRC) con sensorización óptica y de compresión. (31)

El sistema e sensorización óptico se implementa con el sensor de color **TCS34725**, el cual ofrece buena sensibilidad y relación señal-ruido incluso bajo condiciones de iluminación variables. La medición de la presión se realiza mediante un sensor resistivo **FSR UX-402**, capaz de caracterizar la fuerza aplicada sobre el dedo y definir las fases de apoyo, compresión y liberación. Las pruebas experimentales permitieron obtener las curvas de fuerza representativas y determinar los valores de referencia necesarios para asegurar que la presión se mantiene dentro de límites seguros y reproducibles.

Ambos sensores se integran en una estructura compacta que posiciona el dedo de forma estable, manteniendo la alineación entre el sensor óptico y la uña y situando el FSR bajo la yema del dedo. El dispositivo también integra un mecanismo de compresión automatizado basado en el funcionamiento de una leva que actúa sobre un soporte móvil y permite aplicar la presión de manera controlada.

El control y procesamiento de los datos se realiza con la placa **XIAO ESP32-C3**, ideal para su integración en el dispositivo por su tamaño reducido, bajo consumo y capacidad de cálculo suficiente para ejecutar operaciones de preprocesamiento embebido y comunicación IoT. Además, su compatibilidad con baterías de litio proporciona autonomía y portabilidad.

5.1.1 Sensor de color RGB

En este apartado se describen las características técnicas y el principio de funcionamiento del sensor de color **TCS34725**, con el objetivo de comprender cómo realiza la adquisición de las componentes cromáticas y cómo transmite los datos hacia el microcontrolador.

El TCS34725, fabricado por *TAOS (Texas Advanced Optoelectronic Solutions)*, es un sensor óptico capaz de detectar las componentes roja, verde, azul y clara (RGB + Clear) de la luz reflejada por

una superficie. Su arquitectura incluye un filtro de bloqueo infrarrojo que elimina las longitudes de onda fuera del espectro visible para mejorar la fidelidad de la medición y reducir interferencias debidas a fuentes de luz ambiental. Su principio de funcionamiento se muestra en la Figura 12. (32)

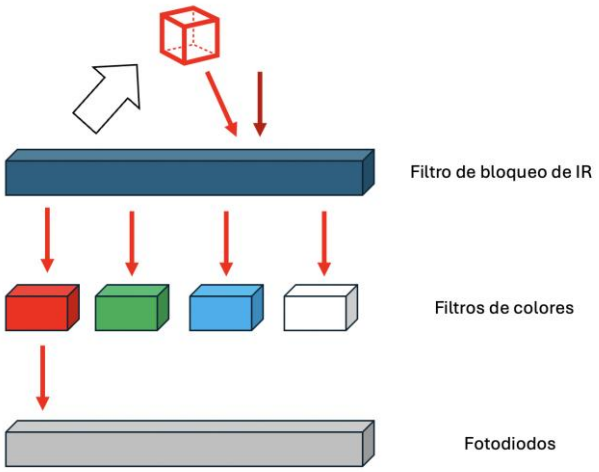


Figura 12: Estructura interna del sensor óptico TCS34725 con filtro de bloqueo IR y matriz de fotodiodos RGB + Clear. (31)

El sensor incorpora una matriz de fotodiodos en celdas filtradas individualmente para los canales R, G, B y Clear. Cada fotodiodo convierte la energía luminosa incidente en una señal eléctrica proporcional a la intensidad del color correspondiente (véase Figura 13). Se obtiene como resultado una lectura de las componentes cromáticas que al combinarse permiten estimar con precisión la composición espectral de la luz reflejada por la uña durante el proceso de reperfusión capilar.

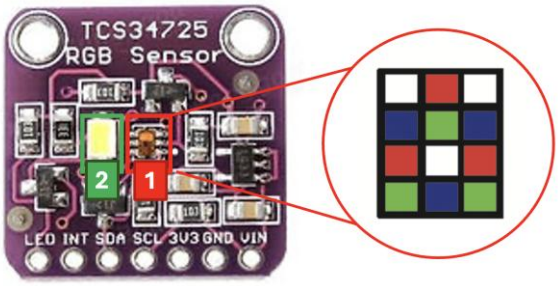


Figura 13: Sensor óptico TCS34725 y disposición de los fotodiodos RGB utilizados para la detección del TRC. (31)

Para garantizar una iluminación homogénea y reproducible, el módulo integra un LED de luz blanca neutra, el cual puede activarse o desactivarse desde el microcontrolador. De esta forma, se mantiene una fuente de luz constante e independiente de las variaciones ambientales, algo esencial para la adquisición fiable de datos ópticos en condiciones clínicas.

Internamente, el TCS34725 incluye convertidores analógico-digitales (ADC) de 16 bits que transforman la señal procedente de los fotodiodos en valores digitales almacenados en registros internos. El tiempo de integración y la ganancia del sensor son configurables, lo que permite ajustar la sensibilidad en función del tipo de tejido a analizar.

La comunicación con el microcontrolador se realiza mediante el protocolo I²C, un estándar ampliamente utilizado en sistemas embebidos por su simplicidad y fiabilidad. Este bus utiliza dos líneas, la SCL (reloj) y SDA (datos), que permiten la transmisión síncrona de información a velocidades de hasta 400 kHz, tal como se muestra en la arquitectura interna del sensor de la Figura 14. El sensor también dispone de una línea de interrupción (INT) que puede activarse cuando la intensidad medida supera umbrales definidos por el usuario, optimizando así el uso de recursos al evitar lecturas continuas innecesarias.

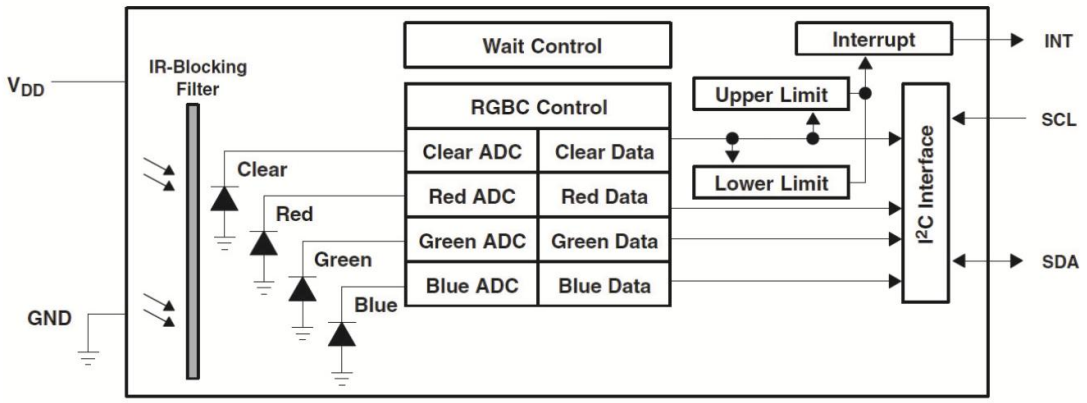


Figura 14: Diagrama funcional interno del sensor óptico TCS34725 y su interfaz de comunicación I²C. (32)

Los pines principales del TCS34725 se presentan en la Figura 15 y son los siguientes:

- **VDD:** alimentación principal del sensor (3,3 V o 5 V según la aplicación).
- **GND:** referencia de tierra del circuito.
- **SCL y SDA:** líneas de comunicación del bus I²C.
- **INT:** salida de interrupción configurable.
- **LED:** control de encendido del diodo emisor incorporado.
- **NC:** pin sin conexión asignada.

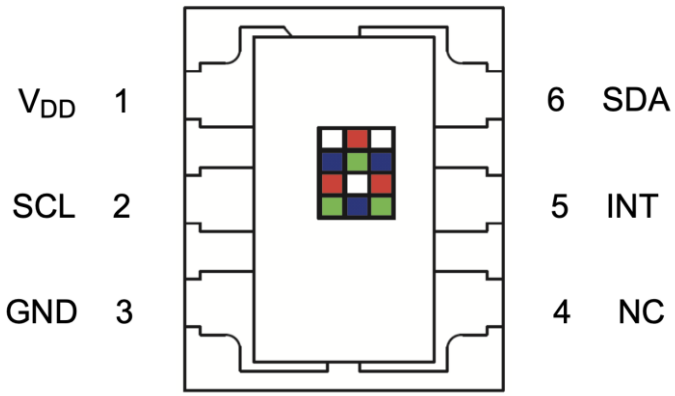


Figura 15: Esquema de pines del sensor óptico TCS34725. (32)

5.1.2 Sensor de fuerza FRS404

En el dispositivo desarrollado, el **FSR** es el encargado de registrar la presión ejercida sobre el dedo durante la medición. Su función es fundamental, ya que permite identificar los momentos de inicio y finalización de la compresión, lo que resulta esencial para sincronizarlo con la lectura del sensor de color. (33)

Los sensores FSR pertenecen a la familia de los dispositivos piezorresistivos, cuya resistencia eléctrica varía en función de la fuerza o presión aplicada sobre su superficie. Este principio fue introducido en la década de los ochenta por Interlink Electronics y se mantiene vigente gracias a su sencillez, fiabilidad y bajo coste de fabricación.

En este proyecto se emplea el modelo **UX-402**, del mismo fabricante, debido a su buena relación entre sensibilidad, estabilidad y facilidad de integración en sistemas portátiles (véase Figura 16). Se trata de un sensor flexible y ultrafino que puede detectar fuerzas a partir de unos 0,1 N y hasta un máximo de 10 N, ideal para el contexto de medición del TRC. Estas características lo hacen idóneo para aplicaciones en las que se requiere una medición de fuerza controlada y repetitiva, como es el caso del procedimiento de evaluación del TRC.

Figura 16: Sensor de fuerza FRS404. (33)

Entre sus ventajas se incluyen su robustez mecánica, con una vida útil superior a millones de ciclos, y su versatilidad dimensional, que permite adaptarlo a distintos formatos o superficies de contacto. Además, su bajo consumo energético y su simplicidad de conexión facilitan su incorporación en dispositivos embebidos alimentados por batería.

5.1.3 Placa de desarrollo XIAO ESP32C3

Dentro del sistema IoT propuesto, la placa de desarrollo constituye el elemento central de control y comunicación, ya que se encarga de gestionar las lecturas de los sensores, realizar el preprocesamiento local de datos y transmitir la información a la infraestructura remota. Para cumplir con estos requisitos, se ha seleccionado la **Seeed Studio XIAO ESP32-C3**, una placa que combina un tamaño reducido, bajo consumo energético y conectividad inalámbrica integrada. Estas son características que la hacen especialmente adecuada para dispositivos biomédicos portátiles. Además, la placa es indicada por el fabricante como ideal para aplicaciones TinyML e IoT, por lo que resulta adecuada para integrar modelos de ML embebido y gestionar la comunicación de datos en tiempo real.

La XIAO ESP32-C3 integra el microcontrolador ESP32-C3 desarrollado por *Espressif Systems*, que está basado en una arquitectura RISC-V de 32 bits y que tiene una frecuencia de operación de hasta 160 MHz. Dispone de 400 KB de SRAM y 4 MB de memoria Flash, por lo que ofrece capacidad suficiente para ejecutar tareas de adquisición, filtrado y transmisión de datos sin requerir hardware adicional. (34)

En cuanto a sus comunicaciones, el módulo incorpora conectividad Wi-Fi (IEEE 802.11 b/g/n) y Bluetooth 5 Low Energy (véase Figura 17), lo que le permite realizar la transmisión inalámbrica de datos tanto hacia plataformas en la nube como hacia dispositivos cercanos. Esta versatilidad facilita su integración en entornos IoMT donde la conectividad estable y de bajo consumo es esencial.

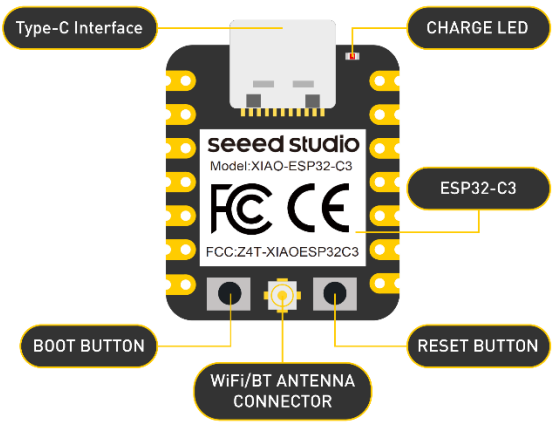


Figura 17: Componentes principales y disposición de la placa de desarrollo Seeed Studio XIAO ESP32-C3. (34)

A nivel de interfaz, la placa cuenta con 11 pines digitales, como se muestra en la Figura 18, configurables como salidas PWM y 4 entradas analógicas utilizables como ADC, además de soportar varios protocolos estándar (UART, I²C, SPI e I²S) que permiten la comunicación directa con sensores externos. También incluye un botón de reinicio, un modo bootloader para programación y un LED indicador integrado.

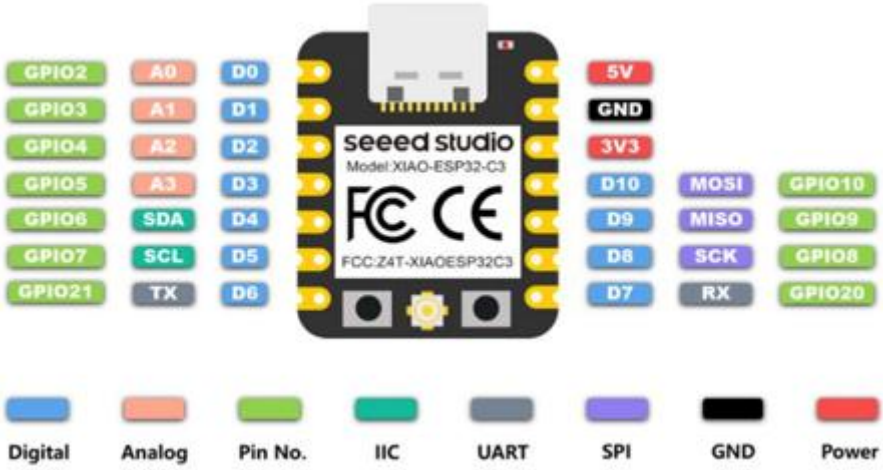


Figura 18: Distribución de pines e interfaces de comunicación de la placa Seeed Studio XIAO ESP32-C3. (34)

En términos energéticos, el consumo de la placa varía según el modo de funcionamiento. En reposo, puede descender por debajo de 44 µA, mientras que en operación normal se mantiene en torno a 25 mA, lo que garantiza una buena eficiencia en dispositivos autónomos. Su tensión de operación es de 5 V cuando se alimenta mediante el conector USB tipo C, aunque también admite una batería de ion-litio de 3,7 V.

5.2 Fundamentos de la arquitectura IoMT y del procesamiento embebido

En este apartado se recopila y analiza la información técnica y conceptual relacionada con los elementos clave de la arquitectura IoMT y del procesamiento embebido. Esta revisión ha servido como base para definir las soluciones tecnológicas empleadas en el sistema desarrollado y para fundamentar las decisiones adoptadas en la fase de implementación.

5.2.1 Concepto y estructura de la IoMT

El IoMT se trata de una extensión del Internet de las Cosas (IoT) aplicada al ámbito sanitario uy que está orientado a conectar dispositivos médicos, sensores fisiológicos y plataformas de gestión de datos dentro de una infraestructura digital integrada. Su principal objetivo es permitir la adquisición, el intercambio y el análisis continuo de información clínica, para favorecer a una atención más preventiva, personalizada y eficiente. (35, 36)

A diferencia del IoT convencional, el IoMT combina elementos físicos y virtuales (pacientes, dispositivos, personal médico y sistemas de información) en un entorno en el que la conectividad y la IA permiten automatizar procesos clínicos y optimizar la toma de decisiones.

En términos estructurales, la IoMT se organiza en capas funcionales interdependientes que garantizan el flujo de información desde los sensores hasta las aplicaciones de supervisión y análisis (35), tal como se ilustra en la Figura 19, donde se muestran los principales niveles. La literatura técnica suele distinguir cuatro niveles principales: sensorización, comunicación, procesamiento y aplicación.

1. **Capa de dispositivos o sensorización:**

   Incluye sensores portátiles, implantables o ambientales que capturan señales fisiológicas como frecuencia cardíaca, saturación de oxígeno o presión arterial. Estos dispositivos incorporan microcontroladores de bajo consumo capaces de realizar un preprocesamiento básico y transmitir la información a nodos superiores mediante tecnologías de comunicación inalámbrica como Bluetooth, ZigBee o Wi-Fi. En entornos clínicos, la elección de Wi-Fi es habitual por su mayor ancho de banda y compatibilidad con infraestructuras hospitalarias.

2. **Capa de comunicación o pasarela:**

   Actúa como enlace entre los dispositivos de sensorización y las plataformas de procesamiento. Aquí se emplean protocolos ligeros y eficientes diseñados para entornos con recursos limitados. En este proyecto, el protocolo **MQTT (Message Queuing Telemetry Transport)** resulta especialmente adecuado, ya que utiliza un modelo publicador–suscriptor que reduce el tráfico de red, minimiza la latencia y garantiza la entrega de mensajes incluso con conectividad inestable. Esta capa puede incorporar también funciones de seguridad y control de acceso, asegurando la confidencialidad e integridad de los datos médicos transmitidos.

3. **Capa de procesamiento o servicios en la nube:**

   En este nivel se almacenan, gestionan y analizan los datos recopilados mediante tecnologías de cloud computing o fog computing, dependiendo de los requisitos de latencia y capacidad. La integración con herramientas de big data y algoritmos de ML permite detectar patrones clínicos, generar alertas tempranas y apoyar la toma de decisiones médicas.

4. **Capa de aplicación o decisión:**

   Es la interfaz final entre el sistema y los usuarios. Aquí se presentan los datos procesados a través de plataformas de visualización o paneles clínicos, facilitando la monitorización en tiempo real, la trazabilidad de los pacientes y la coordinación entre profesionales sanitarios.

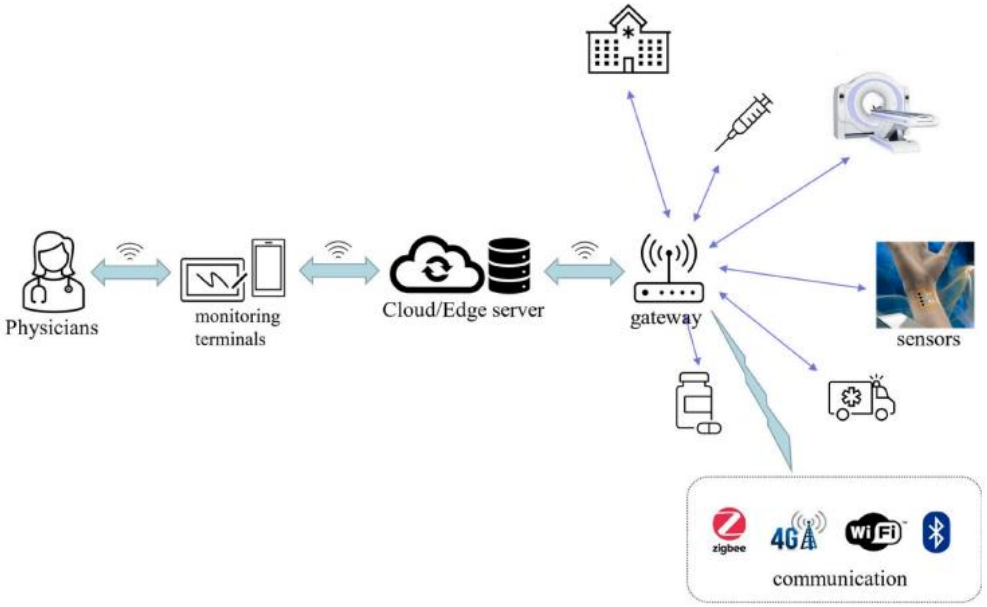


Figura 19: Arquitectura general de un sistema IoMT con comunicación entre sensores, pasarela y entorno de análisis clínico. (35)

El diseño por capas del MT ofrece ventajas como escalabilidad, interoperabilidad y resiliencia frente a fallos de red. Aun dasí, plantea también retos técnicos, especialmente en relación con la seguridad, el consumo energético y la estandarización de protocolos, los cuales son aspectos críticos en entornos donde se manejan datos clínicos sensibles.

En el contexto del presente trabajo, esta arquitectura muestra la base teórica del sistema a desarrollar, en el que la combinación de procesamiento embebido local y comunicación IoMT mediante Wi-Fi y MQTT permite un flujo de datos eficiente, seguro y en tiempo real entre el dispositivo de medición y la infraestructura analítica central.

5.2.2 Fundamentos del procesamiento embebido y edge computing

El procesamiento embebido y el *edge computing* forman la base tecnológica que permite trasladar parte de la inteligencia computacional desde la nube hasta los dispositivos más próximos a la fuente de datos. Esta aproximación busca ofrecer respuestas rápidas, reducir el tráfico en la red y proteger la privacidad de la información, especialmente en entornos con comunicaciones inestables o que manejan datos clínicos sensibles. (37, 38)

En una arquitectura IoMT, los dispositivos embebidos actúan como nodos inteligentes capaces de adquirir, preprocesar y analizar datos antes de enviarlos a niveles superiores de procesamiento. Esta estrategia distribuye la carga computacional entre distintos niveles, lo cuales son el dispositivo (*edge*), los nodos intermedios (*fog*) y la nube, optimizando así el rendimiento global del sistema. En comparación con la computación en la nube, la cual centraliza el procesamiento y puede generar latencias elevadas, el *edge computing* sitúa la toma de decisiones cerca del origen de los datos, lo que resulta esencial en aplicaciones médicas o críticas en tiempo real (38).

La principal ventaja del *edge computing* es su capacidad para reducir la latencia y el ancho de banda requerido, procesando únicamente los datos relevantes y filtrando la información antes de ser enviada. Este enfoque mejora la eficiencia energética y permite mantener un control local sobre los datos, reforzando la privacidad y la seguridad del paciente. Además, posibilita el funcionamiento autónomo de los dispositivos en escenarios donde la conexión a Internet no es continua, como ocurre en muchas unidades clínicas portátiles. (37, 38)

El avance de los microcontroladores modernos, como el ESP32-C3, ha facilitado el despliegue de sistemas embebidos con capacidad de procesamiento suficiente para ejecutar modelos de inferencia local. Estos dispositivos combinan bajo consumo energético, conectividad inalámbrica (Wi-Fi y Bluetooth LE) y compatibilidad con bibliotecas de ML optimizadas, lo que los hace idóneos para sistemas biomédicos portátiles, como se representa en la Figura 20, donde se muestra un esquema de la arquitectura típica de un nodo embebido con capacidades de ML local.

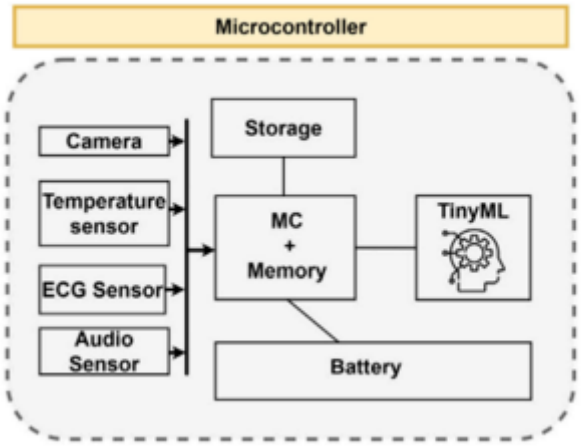


Figura 20: Arquitectura general de un sistema embebido con sensorización y procesamiento local mediante TinyML. (37)

La integración de TinyML representa una evolución clave dentro del *edge computing*. Esta tecnología permite ejecutar modelos de *ML* directamente en microcontroladores con recursos limitados, mediante técnicas de optimización como cuantización, poda y distilación del conocimiento, que reducen el tamaño del modelo sin comprometer de forma significativa su precisión (39). De este modo, las tareas como la clasificación de la calidad de señal o la detección de eventos fisiológicos pueden realizarse en el propio dispositivo, evitando la dependencia de la nube y garantizando una respuesta inmediata.

Estudios recientes señalan que los modelos TinyML bien optimizados pueden reducir su huella de memoria más de un 90 %, alcanzando latencias inferiores a 10 ms y consumos energéticos de apenas 25 mW, cifras que permiten mantener dispositivos "siempre activos" con alimentación por batería (39). Esta capacidad ha fomentado su aplicación en sistemas de monitorización médica, donde el tiempo de respuesta y la privacidad son prioritarios.

Por otro lado, la combinación de *edge* y *cloud computing* da lugar a arquitecturas híbridas o de tipo *fog*, que permiten equilibrar la capacidad de cómputo local con el almacenamiento y análisis avanzado en la nube. Estas arquitecturas aprovechan la proximidad del procesamiento local para gestionar tareas sensibles al tiempo y delegan el entrenamiento o la agregación de datos históricos al entorno cloud, logrando así un equilibrio entre latencia, escalabilidad y consumo energético (38).

Desde un punto de vista práctico, esta filosofía de diseño es la que sustenta el sistema propuesto en el presente trabajo. El microcontrolador realiza un preprocesamiento local y la inferencia embebida de la señal óptica mediante un modelo TinyML, mientras que los datos estructurados se transmiten mediante el protocolo MQTT a un entorno de análisis distribuido. Este enfoque combina la inmediatez del procesamiento embebido con la capacidad de almacenamiento y análisis del sistema central, constituyendo una solución eficiente, segura y coherente con las tendencias actuales de la ingeniería biomédica conectada.

5.2.3 Principios de integración IoMT–Big Data

La integración entre el IoMT y las tecnologías de Big Data representan uno de los pilares más sólidos en la evolución de los sistemas inteligentes de monitorización sanitaria. En este contexto, los dispositivos biomédicos conectados actúan como fuentes de información continua y generan grandes volúmenes de datos heterogéneos que, tras ser adquiridos y preprocesados, se transmiten a entornos de almacenamiento y análisis distribuidos. Este flujo de información crea una infraestructura capaz de transformar señales fisiológicas brutas en conocimiento clínico útil y accionable en tiempo real. (36, 40, 41)

De acuerdo con Ahila et al. (40), las arquitecturas IoMT modernas combinan sensores biomédicos, pasarelas de comunicación (Wi-Fi, Bluetooth o MQTT) y servicios en la nube para construir un entorno de *telemonitorización inteligente*. En estos sistemas, la capa de sensorización recoge variables como la saturación de oxígeno, la frecuencia cardíaca o la temperatura, mientras que las pasarelas locales gestionan la transmisión segura de los datos hacia una nube médica. Allí, los registros son depurados y organizados mediante técnicas de preprocesamiento como eliminación de ruido, normalización y tratamiento de valores perdidos para facilitar su análisis posterior. Este enfoque reduce la carga sobre los profesionales de la salud y mejora la continuidad del cuidado, ya que las alertas o diagnósticos preliminares pueden generarse automáticamente mediante algoritmos de ML.

Niu et al. (36) destacan que esta integración se apoya en una arquitectura multinivel, donde los nodos IoMT forman el nivel más cercano al paciente, seguidos por una capa intermedia de procesamiento (*edge/fog computing*) y un nivel superior de análisis en la nube. Este modelo híbrido distribuye la inteligencia computacional, ya que el dispositivo realiza tareas de filtrado y clasificación local, mientras que la nube agrega y correlaciona los datos a gran escala. Este esquema jerárquico se ilustra en la Figura 21, donde se representa la organización por capas típica de un sistema IoMT–Big Data, desde los dispositivos de sensorización hasta la capa de análisis en la nube y la interacción con los profesionales sanitarios. Gracias a ello, se logra una mayor escalabilidad, reducción de la latencia y un uso más eficiente del ancho de banda. La sincronización entre ambos niveles permite, además, la generación de modelos predictivos actualizados de forma continua con los nuevos datos adquiridos.

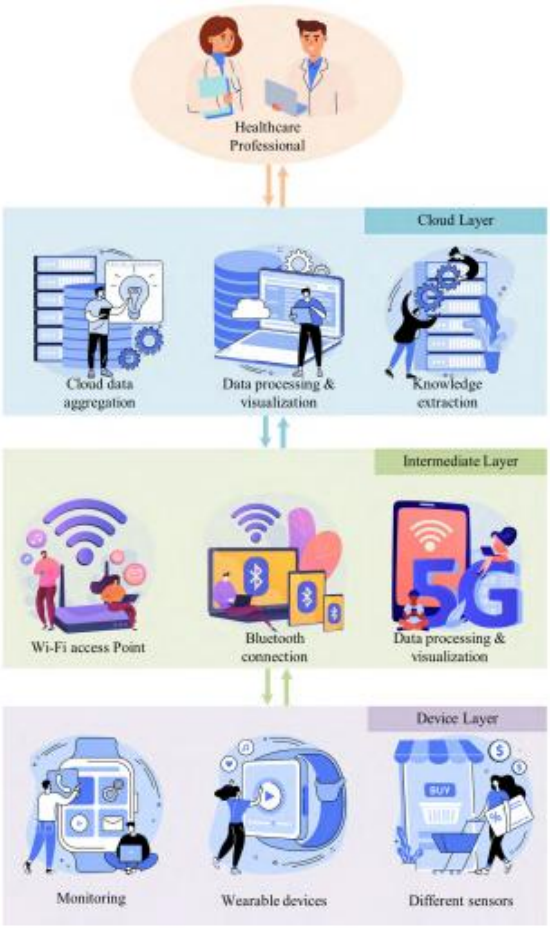


Figura 21: Arquitectura por capas de un sistema IoMT–Big Data que integra sensorización, comunicación y análisis clínico en la nube. (36)

Por su parte, Awrahman et al. (41) subrayan que el Big Data aplicado a la salud se caracteriza por las "4V": volumen, velocidad, variedad y veracidad, lo que exige infraestructuras capaces de gestionar datos provenientes de fuentes heterogéneas (EHR, sensores, imágenes médicas, redes sociales o dispositivos portátiles). En este contexto, tecnologías como Hadoop, Spark o HBase se emplean para procesar grandes conjuntos de datos clínicos y generar conocimiento de valor. Estas plataformas distribuidas no solo permiten almacenar y analizar la información médica en tiempo real, sino también asegurar la trazabilidad de los datos y su disponibilidad para futuras investigaciones o entrenamiento de modelos de IA. Asimismo, los autores destacan la relevancia de los mecanismos de seguridad y privacidad en entornos de datos distribuidos, especialmente cuando la información procede de múltiples instituciones médicas o de pacientes conectados remotamente.

La sinergia entre IoMT y Big Data facilita la creación de ecosistemas inteligentes de salud que sean capaces de integrar mediciones fisiológicas, historiales clínicos y análisis computacionales avanzados dentro de una misma infraestructura digital. En estos entornos, el procesamiento

local de señales (por ejemplo, mediante modelos TinyML) puede complementar el análisis global de patrones en la nube, lo que permite tanto la detección inmediata de anomalías como la evaluación longitudinal del estado del paciente. Este esquema híbrido es particularmente útil en contextos críticos, como la monitorización en UCI o la atención domiciliaria, donde la inmediatez y la fiabilidad de la información son esenciales.

5.2.4 Ventajas y desafíos en entornos clínicos

La incorporación del IoMT en la práctica clínica supone una transformación importante en la gestión, el seguimiento y la toma de decisiones en salud. La integración de sensores biomédicos, sistemas embebidos y plataformas de análisis distribuido ha permitido avanzar hacia modelos de atención más personalizados, preventivos y continuos, lo que se enmarca dentro del paradigma de la llamada *Healthcare 4.0* (42).Estos avances han demostrado su capacidad para mejorar la eficiencia del personal sanitario, reducir costes operativos y, sobre todo, aumentar la seguridad y el bienestar de los pacientes mediante la monitorización continua y el diagnóstico temprano.

Entre las principales ventajas del uso del IoMT en entornos clínicos destacan la mejora de la precisión diagnóstica y la reducción de tiempos de respuesta. Los dispositivos conectados pueden registrar parámetros fisiológicos en tiempo real y enviarlos automáticamente a sistemas de análisis que pueden generar alertas ante desviaciones significativas. De este modo, los clínicos disponen de información inmediata y contextualizada, lo que facilita la toma de decisiones basada en datos objetivos. Además, el acceso remoto a la información posibilita un seguimiento continuo del paciente incluso fuera del hospital, reduciendo la necesidad de visitas presenciales y optimizando los recursos asistenciales (43).

Otra ventaja relevante es la optimización del flujo de trabajo clínico. La digitalización de procesos mediante plataformas IoMT y sistemas de registro electrónico de salud (EHR) interconectados permite una gestión más ágil y coordinada entre departamentos, disminuyendo los errores humanos y mejorando la trazabilidad de la información. Asimismo, la incorporación de IA y ML en el procesamiento de datos médicos favorece la detección temprana de patrones anómalos, como la predicción de infecciones o la identificación de deterioros hemodinámicos antes de que sean clínicamente evidentes. En el caso de sistemas embebidos como el desarrollado en este trabajo, la posibilidad de integrar algoritmos de clasificación directamente en el dispositivo reduce la latencia y aumenta la autonomía operativa, los cuales son factores clave en entornos críticos como las UCI.

No obstante, estos beneficios traen consigo también con una serie de desafíos técnicos y operativos que condicionan la adopción generalizada del IoMT en la práctica clínica. En primer lugar, se identifican las limitaciones energéticas y de fiabilidad de los dispositivos. Muchos sensores y nodos embebidos dependen de baterías de corta duración, lo que dificulta su uso prolongado y continuo en pacientes hospitalizados o en monitorización domiciliaria. Además,

las interrupciones en la conectividad inalámbrica (Wi-Fi, Bluetooth, MQTT) pueden afectar la transmisión de datos en tiempo real, generando lagunas o inconsistencias en los registros.

En segundo lugar, la seguridad y privacidad de los datos médicos continúa siendo un aspecto crítico. La naturaleza sensible de la información sanitaria exige el cumplimiento estricto de normativas como el RGPD, lo que implica la aplicación de mecanismos de cifrado, autenticación y control de acceso robustos. Soluciones emergentes como el uso de blockchain o el cifrado homomórfico, mencionadas por Al Khatib et al. (43), ofrecen vías prometedoras para garantizar la integridad y trazabilidad de los datos sin comprometer la usabilidad del sistema.

También persisten retos de interoperabilidad y estandarización. Como subrayan Osama et al. (42), la coexistencia de múltiples fabricantes, protocolos de comunicación y formatos de datos limita la integración entre sistemas clínicos heterogéneos. La adopción de estándares abiertos como HL7 FHIR o IEEE 11073 es indispensable para asegurar la compatibilidad entre plataformas y favorecer la construcción de ecosistemas sanitarios conectados.

Finalmente, en el plano organizativo el uso del IoMT plantea desafíos relacionados con la aceptación por parte del personal clínico. La incorporación de nuevas tecnologías requiere procesos de formación, adaptación de rutinas y confianza en los sistemas automatizados. Asimismo, la sobrecarga informativa derivada del flujo continuo de datos puede dificultar la interpretación clínica si no se cuenta con herramientas de visualización y análisis adecuadas. En este sentido, la integración de paneles interactivos y sistemas de visualización en tiempo real, como los implementados en este proyecto, contribuye a mejorar la comprensión y el uso práctico de los datos médicos.

Capítulo 6. DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN PROPUESTA

Este capítulo recoge el desarrollo técnico del sistema propuesto, abarcando la implementación completa desde la adquisición de datos en el dispositivo IoT hasta su procesamiento y visualización en tiempo real.

Se estructura en cuatro bloques principales: la infraestructura de comunicación IoT basada en la placa Seeed XIAO ESP32-C3 y el protocolo MQTT; el pipeline de ingesta y procesamiento en tiempo real mediante Spark Structured Streaming y bases de datos PostgreSQL/Parquet; el módulo de clasificación con técnicas de *ML*; y el entorno de visualización en Grafana para la supervisión clínica y técnica del sistema.

El desarrollo técnico descrito en este capítulo se fundamenta en los conceptos y tecnologías analizados en el Capítulo 5 , donde se revisan las arquitecturas IoMT, los principios del procesamiento embebido y las estrategias de integración del ML en dispositivos de bajo consumo. A partir de estos fundamentos, se plantea una solución estructurada en distintas capas funcionales que reproducen la arquitectura ideal identificada en el marco teórico: una capa embebida responsable de la adquisición de datos y la inferencia local mediante un modelo TinyML, una capa de comunicación IoT basada en mensajería MQTT que permite la transmisión continua y estructurada de la información y una capa de procesamiento y visualización orientada a la gestión y análisis de los datos clínicos en tiempo real. Esta organización, junto con el uso de tecnologías Big Data y herramientas de análisis distribuido, permite garantizar la trazabilidad, escalabilidad e interoperabilidad del sistema dentro de entornos clínicos conectados. De este modo, el diseño e implementación que se presentan a continuación responden directamente a los principios teóricos presentados en el capítulo anterior para consolidar así una solución coherente y alineada con los requisitos funcionales de una arquitectura IoMT para la monitorización del TRC.

6.1 Diseño e implementación de la infraestructura IoT (ESP32-C3 y protocolo MQTT)

Este apartado describe el diseño e implementación de la infraestructura de comunicación IoT, la cual constituye el punto de partida de todo el sistema. Su objetivo es permitir que el dispositivo físico, formado por la placa Seeed XIAO ESP32-C3 junto con el sistema de sensorización, se conecte a la red y transmita los datos al ecosistema de procesamiento en tiempo real.

En la Figura 22 se representa el flujo de comunicación del sistema, en el que el dispositivo basado en la placa ESP32-C3 actúa como cliente MQTT y envía los datos generados hacia un *broker* Mosquitto, que los distribuye al entorno de procesamiento en tiempo real implementado con *Spark Structured Streaming*.

Figura 22: Esquema de la infraestructura de comunicación IoT mediante protocolo MQTT entre el dispositivo ESP32-C3, el broker Mosquitto y el sistema de procesamiento en tiempo real. Fuente: elaboración propia.

Este diseño se apoya en tecnologías abiertas y ligeras, como MQTT para la comunicación IoT y Spark Structured Streaming para el procesamiento, para poder garantizar una infraestructura modular, escalable y trazable entre el dispositivo y la capa de análisis.

Para habilitar la comunicación, se configura el microcontrolador en el entorno Arduino IDE 2.3 y se emplea el protocolo MQTT, que como ya se ha presentado en el Capítulo 5, es un estándar ligero y eficiente ampliamente utilizado en entornos IoT. Este protocolo establece un modelo publish/subscribe, en el que los dispositivos publican información en un broker, que a su vez la distribuye a los suscriptores interesados.

En este contexto, el presente apartado aborda cuatro etapas:

1. **Conexión Wi-Fi** de la placa ESP32-C3 a una red de 2.4 GHz mediante la librería *WiFi.h*.
2. **Implementación del cliente MQTT básico**, encargado de enviar telemetría periódica.
3. **Extensión del cliente MQTT con jerarquía de topics y control LWT**, que dota al sistema de bidireccionalidad y robustez.
4. **Validación extremo a extremo**, en la que se comprueba la estabilidad, la reconexión automática y la latencia del flujo completo.

Esta infraestructura constituye el enlace entre la capa física de adquisición y la plataforma de procesamiento en streaming, y garantiza una transmisión continua para su integración con Spark Structured Streaming en la siguiente fase del proyecto.

6.1.1 Conexión de la placa Seeed XIAO ESP32-C3 a la red Wi-Fi

El primer paso en el desarrollo de la infraestructura IoT consiste en conectar la placa Seeed XIAO ESP32-C3 a una red Wi-Fi. Este paso es imprescindible para que el dispositivo pueda enviar y recibir datos a través de Internet y, posteriormente, integrarse con el broker MQTT y con el *pipeline* Big Data.

La programación del microcontrolador se realiza sobre el entorno Arduino IDE 2.3 y el *core* oficial esp32, utilizando la librería estándar WiFi.h desarrollada por Espressif Systems). Esta librería

está incluida en el *Arduino oficial core para ESP32* y utiliza las APIs Wi-Fi descritas en la guía del ESP-IDF (43, 44).

Es importante mencionar que la familia de placas ESP32-C3 solo es compatible con redes IEEE 802.11 b/g/n de 2.4 GHz, lo cual está especificado en la documentación oficial del fabricante Seeed Studio (34), por lo que se deben descartar automáticamente las redes de 5 GHz. Antes de establecer la conexión con la red WiFi, se realiza un escaneo de las redes disponibles para comprobar que la red objetivo se encuentra en la banda adecuada para poder conectar la placa de desarrollo.

El código completo del procedimiento se incluye en el Anexo A, en el que se muestra la secuencia de conexión implementada. Al finalizar el escaneo, se pueden obtener los principales parámetros de cada red detectada, como el SSID, la intensidad de señal (RSSI), el canal de transmisión y el tipo de seguridad. Esta información es suficiente para identificar la red objetivo y evaluar la calidad de la conexión. Conviene tener en cuenta que este tipo de escaneo bloquea temporalmente la CPU del microcontrolador mientras se ejecuta, lo que en entornos concurrentes podría afectar a otras tareas temporizadas. Aun así, esto no representa un problema en esta fase de arranque.

Ejemplo de salida del monitor serie:

Encontradas 3 redes:

SSID RSSI Canal Banda Seguridad

---

* MOVISTAR_XXX -55 2 2.4G WPA2

MiRed5G -48 36 5G* WPA2/WPA3

Invitados -72 11 2.4G WPA2

(* = red objetivo; 5G* = 5 GHz, no compatible con ESP32-C3)

En este caso, la red *MOVISTAR_XXX, que ha sido la utilizada en las posteriores conexiones,* presenta buena potencia de señal (–55 dBm), canal 2 y seguridad WPA2, confirmando que es una red válida de 2.4 GHz para conectar el dispositivo. Esta validación previa evita errores de conexión posteriores en la fase de publicación MQTT, en las que la estabilidad de la red es un factor crítico. Además, proporciona métricas de entorno (intensidad de señal, canal, cifrado) útiles para documentar las condiciones de conectividad.

6.1.2 Implementación del cliente MQTT básico

Una vez establecida la conexión Wi-Fi, el siguiente paso consiste en habilitar un mecanismo de comunicación entre el dispositivo IoT y el resto de la infraestructura de datos. Para ello se utiliza el protocolo MQTT como se ha afirmado previamente.

En este nivel básico, la ESP32-C3 actúa como un cliente MQTT que se conecta a un *broker*, un servidor intermedio que es el encargado de recibir los mensajes publicados por unos dispositivos y distribuirlos a los que estén suscritos al mismo *topic*. Este modelo se denomina publish/subscribe (pub/sub), ya que el dispositivo publica información en un *topic* (por ejemplo el empleado en este proyecto tfm_clara_demo/status) y otros sistemas, como un PC con Mosquitto o, más adelante, Spark Streaming, pueden suscribirse a ese *topic* para recibir los datos en tiempo real. La Figura 23 muestra la estructura y principio de funcionamiento de este protocolo de comunicación:

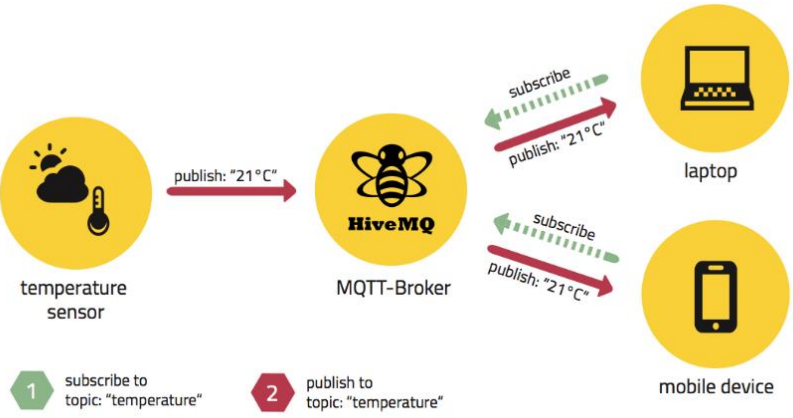


Figura 23: Esquema de comunicación MQTT en arquitectura publish/suscribe. (43)

Durante la validación técnica se utilizó un broker público (test.mosquitto.org) únicamente con datos simulados y sin información sensible. Esto permite validar la comunicación sin necesidad de desplegar todavía un *broker* local. En un entorno clínico real, este componente se sustituiría por un broker privado autenticado y con comunicación cifrada para poder garantizar la integridad y confidencialidad del intercambio de datos.

Al no requerir autenticación, únicamente se publican datos no sensibles para la validación. Es importante mencionar que al tratarse de un entorno abierto, las comunicaciones no están cifradas y los mensajes no se retienen, por lo que puede haber limitaciones de disponibilidad o interferencia con otras conexiones. No obstante, para esta fase inicial de pruebas, es adecuado al permitir una validación rápida y reproducible del flujo MQTT. El código empleado mostrado en el Anexo B se basa en la librería PubSubClient.

El flujo general es el siguiente:

1. La ESP32-C3 inicializa un objeto cliente (WiFiClient net; PubSubClient mqtt(net);) después de conectarse a la WiFi y define un identificador único que se utiliza en el broker.
2. Después se configura el servidor MQTT (mqtt.setServer(MQTT_HOST, MQTT_PORT)) y la función de callback (mqtt.setCallback(mqttCallback)) que se activa al recibir mensajes.

3. En la función loop() el programa comprueba continuamente que la conexión se mantiene (ensureMqtt()), procesa mensajes pendientes con mqtt.loop() y publica un JSON cada 5 segundos en el topic tfm_clara_demo/status. Este intervalo se define únicamente para la validación y puede ajustarse posteriormente según la frecuencia de adquisición del sensor.

El payload publicado incluye dos parámetros básicos de telemetría: la dirección IP asignada (WiFi.localIP()) y la intensidad de señal (WiFi.RSSI()). Un ejemplo de mensaje es:

{"ip":"192.XXX.XX.XX","rssi":-55}

De este modo, se valida que el dispositivo no solo se conecta correctamente a la red y al broker, sino que también transmite información en tiempo real.

Para comprobar el funcionamiento se utilizaron dos salidas complementarias.

En el Monitor Serie de Arduino IDE, la placa reporta el estado de cada publicación con mensajes como el de la Figura 24:

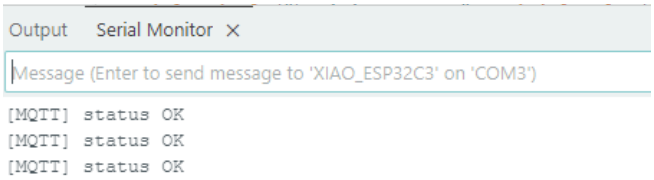


Figura 24: Confirmación de publicación MQTT en el monitor serie de Arduino IDE. Fuente: Elaboración propia.

Este mensaje confirma que la librería ha enviado correctamente el paquete al broker y que la conexión MQTT está estable. En paralelo, en un PC con Windows se instaló la herramienta Eclipse Mosquitto que proporciona los comandos mosquitto_sub y mosquitto_pub. Al ejecutar desde la consola el comando:

mosquitto_sub -h test.mosquitto.org -t "tfm_clara_demo/status" -v

se visualizan en tiempo real los mensajes JSON publicados por la ESP32-C3 (ver Figura 25).

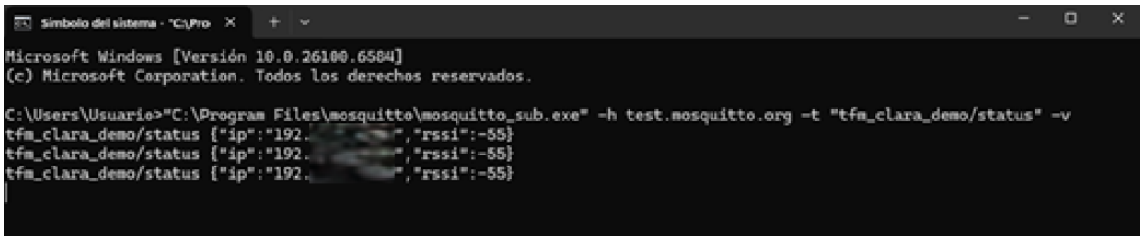


Figura 25: Recepción de mensajes MQTT con Mosquitto. Fuente: elaboración propia.

De esta forma, se demuestra el funcionamiento extremo a extremo de la comunicación, ya que la placa publica, el broker intermedio distribuye y el cliente recibe la información.

Este hito inicial es fundamental en el desarrollo del sistema, ya que constituye el puente entre la capa IoT encargada de recibir y enviar datos mediante la ESP32-C3 y la capa de procesamiento Big Data que posteriormente recibirá esta información para su análisis en tiempo real.

Al igual que en el caso de la conexión Wi-Fi, el desarrollo de este código se basa en bibliotecas oficiales ampliamente documentadas como WiFi.h y PubSubClient dentro del core de Arduino-ESP32. Para la verificación del flujo se empleó la implementación oficial de Eclipse Mosquitto como referencia. (43, 46, 47)

6.1.3 Cliente MQTT con topics jerárquicos y control LWT

Sobre la base de la publicación periódica en MQTT, se amplía el cliente de la ESP32-C3 para proporcionarle bidireccionalidad y robustez. Para ello se adopta un *namespace* único, en este caso tfm_clara_demo/ que funciona como prefijo común para los *topics* y agrupa todos los mensajes del proyecto. Esto facilita la organización de la comunicación y permite escalar el sistema a varios dispositivos o servicios sin riesgo de colisiones entre *topics*.

Dentro de este *namespace* se definen *topics* con distintas funciones:

- tfm_clara_demo/status para la telemetría
- tfm_clara_demo/cmd para los comandos entrantes
- tfm_clara_demo/ack para las respuestas del dispositivo
- tfm_clara_demo/lwt que constituye el estado (Last Will and Testament)

El esquema anteriormente mostrado en la Figura 24 muestra la jerarquía de topics y la dirección de comunicación entre el cliente, el broker y el suscriptor.

Después de conectar la Wi-Fi, el programa inicializa el cliente MQTT (WiFiClient + PubSubClient), configura el servidor (mqtt.setServer(MQTT_HOST, MQTT_PORT)) y registra el *callback* de recepción (mqtt.setCallback(mqttCallback)). Al establecer sesión con el *broker* se fija un LWT. Si el dispositivo se desconecta de forma inesperada, el *broker* publica automáticamente offline en tfm_clara_demo/lwt. El mensaje LWT (Last Will and Testament) es una característica nativa del protocolo MQTT que permite notificar de desconexiones y mantener la coherencia del estado del sistema.

Una vez conectada, la placa publica online con la marca retain para que cualquier suscriptor nuevo vea el último estado de forma inmediata. A continuación, el dispositivo se suscribe a tfm_clara_demo/cmd para recibir órdenes remotas.

La función mqttCallback(...) procesa el contenido recibido en cmd. Aunque MQTT transmite los mensajes como secuencias de bytes, aquí se utiliza el formato JSON por su legibilidad y por facilitar el tratamiento que se realizará posteriormente ([Eclipse Foundation, 2019]).

El contenido del mensaje se convierte a mayúsculas y se gestionan tres comandos principales:

- **PING**, que genera la respuesta {"ack":"PONG","ms":<millis>} publicada en ack

- **LED_ON** y **LED_OFF**, que activan o desactivan el LED integrado de la placa (LED_BUILTIN) y confirman con {"ack":"LED_ON"} / {"ack":"LED_OFF"}.

En el bucle (loop()), se mantiene la conexión (mqtt.loop()), se comprueba la reconexión (mqttConnect()) y se publica telemetría ampliada cada 5 segundos en status. El JSON incluye la IP local, el RSSI, un contador tick y una lectura analógica a0.

Una vez implementada la lógica del cliente, se procede a validar su funcionamiento con el *broker* público mediante las herramientas de consola de Mosquitto. Esta se realiza con Eclipse Mosquitto instalado en el PC, el cual incluye los clientes de consola mosquitto_sub y mosquitto_pub. Al suscribirse al *namespace* completo con:

> "C:\Program Files\mosquitto\mosquitto_sub.exe" -h test.mosquitto.org -t "tfm_clara_demo/#" -v

se observa el siguiente flujo de mensajes (ver Figura 26):

- El topic tfm_clara_demo/lwt publica *online* al iniciar el dispositivo
- La telemetría periódica aparece en tfm_clara_demo/status {...}
- Los comandos entrantes se registran en tfm_clara_demo/cmd
- Las respuestas se devuelven en tfm_clara_demo/ack

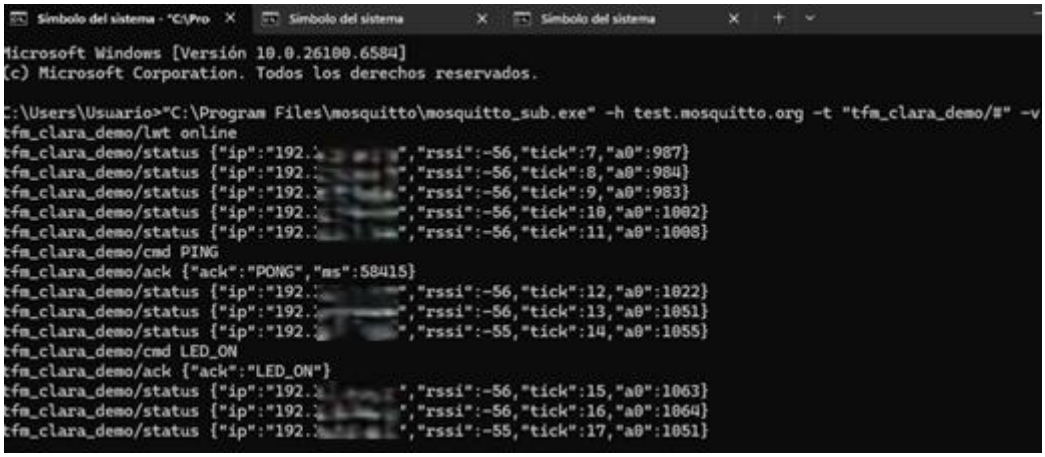


Figura 26: Mensajes del namespace tfm_clara_demo/# con telemetría, comandos y respuestas en tiempo real. Fuente: elaboración propia.

El envío de órdenes se realiza desde otra terminal con los comandos (ver Figuras 27 y 28):

> "C:\Program Files\mosquitto\mosquitto_pub.exe" -h test.mosquitto.org -t "tfm_clara_demo/cmd"

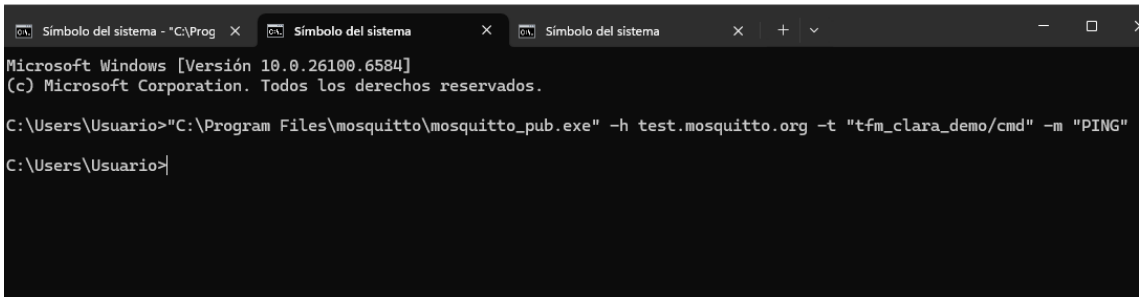


Figura 27: Publicación del comando PING en el topic tfm_clara_demo/cmd. Fuente: elaboración propia.

y

"C:\Program Files\mosquitto\mosquitto_pub.exe" -h test.mosquitto.org -t "tfm_clara_demo/cmd" -m "LED_ON"

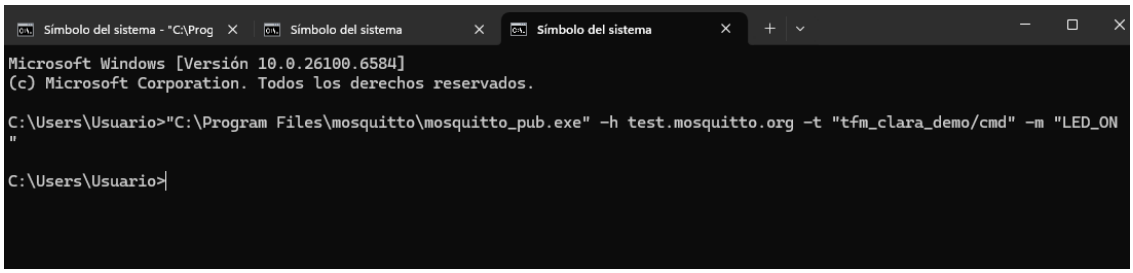


Figura 28: Publicación del comando LED_ON para control del LED integrado. Fuente: elaboración propia.

Cuando la suscripción se realiza sobre tfm_clara_demo/#, también aparecen los propios comandos, ya que el *broker* muestra todos los mensajes del árbol de *topics*. Los acks confirman que el *callback* del firmware se ejecuta correctamente y que el flujo entre el bróker y el dispositivo funciona de forma bidireccional. Si el LED físico no se enciende, por ejemplo por diferencias de pin según placa, la confirmación en ack sigue siendo una evidencia válida.

Este diseño alineado con el modelo *publish/subscribe* del estándar MQTT, ofrece un canal para la telemetría, el control remoto y la supervisión del estado del dispositivo. Además, deja preparado el sistema para que las herramientas de ingestión que se usarán en la siguiente fase de desarrollo (Spark Structured Streaming) se suscriban al status y procesen los mensajes en tiempo real.

Esta implementación realizada se fundamenta en bibliotecas y documentación de referencia. (48)

6.1.4 Validación extremo a extremo

La validación extremo a extremo del cliente MQTT desarrollado para la ESP32-C3 tiene como objetivo comprobar que el dispositivo no solo publica telemetría periódica, sino que también gestiona correctamente su estado de conexión, responde a comandos remotos con confirmación y se recupera automáticamente ante caídas de red.

Durante las primeras pruebas se detectó que el sistema no reflejaba adecuadamente el paso a *offline* al desconectar la placa, por lo que otros clientes no podrían saber su estado real. Por ello, se aplicaron varias modificaciones al firmware (recogidas en el Anexo C):

1. **LWT con retención (retain)**: se marca el mensaje *Last Will and Testament* como retain dentro de la función mqtt.connect(...) para que el *broker* conserve el último valor publicado en el *topic* de estado (online u offline) y lo entregue a cualquier nuevo suscriptor, incluso aunque la placa ya esté desconectada.
2. **Reducción del intervalo *keepAlive***: se configura mqtt.setKeepAlive(5) para que el cliente envíe *PINGs* cada cinco segundos. Así, el *broker* detecta desconexiones en menos de diez segundos y publica *offline* casi de inmediato si la placa deja de responder.
3. **Publicaciones en status con retain**: los mensajes de telemetría se publican también con retención (mqtt.publish(..., true)) para que un nuevo suscriptor pueda conocer el último valor sin tener que esperar a la siguiente publicación. Esto facilita el arranque de consumidores como **Spark Streaming** que reciben un dato inicial inmediato.

Estas modificaciones permiten que el sistema refleje el estado del dispositivo y mantenga la sesión activa de manera robusta. Al suscribirse al *topic* tfm_clara_demo/lwt, se observan las transiciones online -> offline -> online al conectar, desconectar y volver a encender la placa (Figura 29).

El uso de retain asegura que los nuevos suscriptores vean el último estado publicado, mientras que el intervalo reducido de *keepAlive* acelera la detección de caídas. Cuando la conexión se restablece, el firmware ejecuta automáticamente mqttConnect(), restablece la sesión y reanuda la telemetría en status sin necesidad de reiniciar la ESP32-C3.

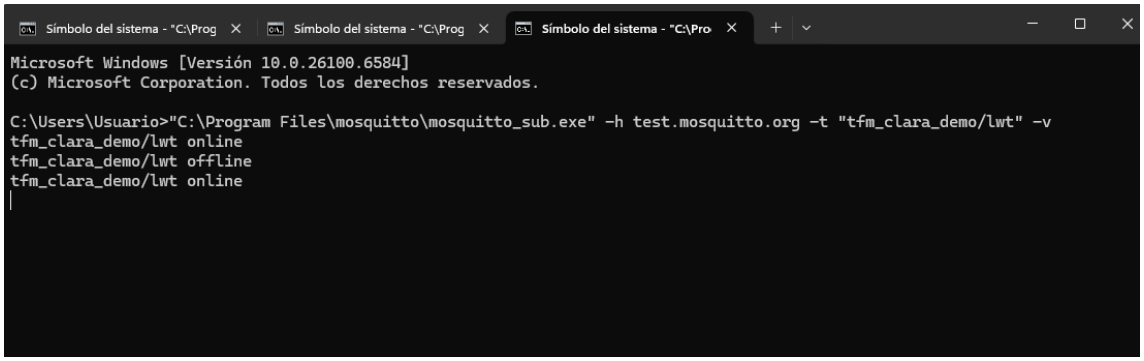


Figura 29: Suscripción al topic tfm_clara_demo/lwt.

En paralelo, se valida la publicación periódica de telemetría en el *topic* tfm_clara_demo/status. Cada cinco segundos, la placa genera un mensaje en formato JSON con los siguientes campos:

- **ip**: dirección IP local asignada
- **rssi**: intensidad de señal Wi-Fi en dBm

- **tick**: contador incremental
- **a0**: lectura analógica

Los mensajes se registran mediante mosquitto_sub.exe (ver Figura 30), que redirige la salida a un archivo de log para poder analizarlos más adelante:

```
C:\Program Files\mosquitto\mosquitto_sub.exe -h test.mosquitto.org -t "tfm_clara_demo/status" -v > C:\Users\Usuario\Desktop\status_log.txt
```

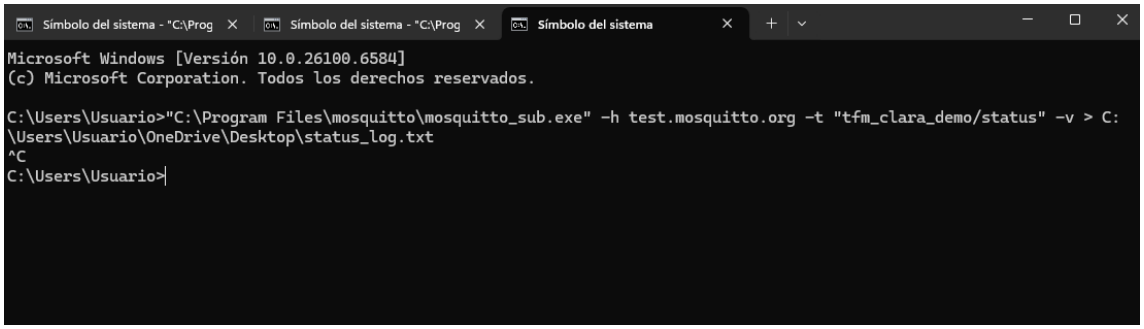


Figura 30: Ejecución de mosquitto_sub.exe con redirección de salida para almacenar los mensajes publicados en tfm_clara_demo/status. Fuente: elaboración propia.

El archivo generado (status_log.txt) que se muestra en la Figura 31 contiene una secuencia continua de registros JSON, y confirma tanto la periodicidad como la persistencia de los datos para su posterior ingesta.

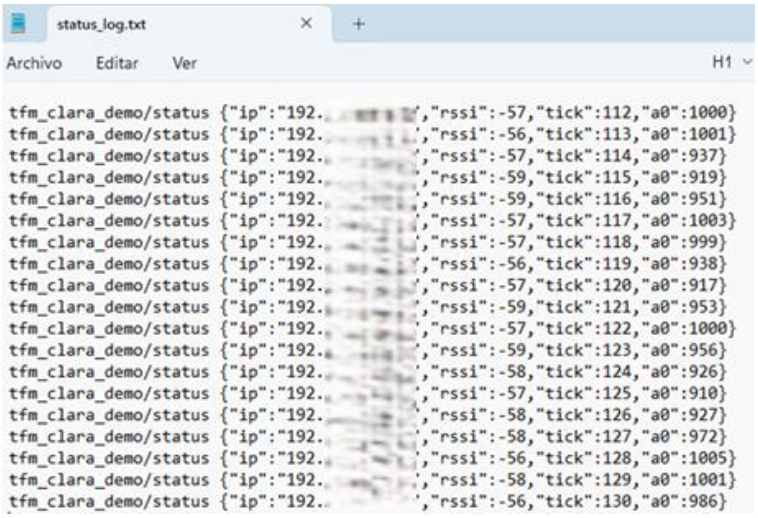


Figura 31: Contenido del archivo status_log.txt. Fuente: elaboración propia.

Otro aspecto fundamental de la validación consiste en comprobar la bidireccionalidad del sistema, es decir, que la ESP32-C3 no solo publique información, sino que también reciba y ejecute comandos enviados desde otro cliente MQTT.

Para ello, se publican mensajes en el *topic* tfm_clara_demo/cmd utilizando el comando mosquitto_pub.exe:

C:\Program Files\mosquitto\mosquitto_pub.exe -h test.mosquitto.org -t "tfm_clara_demo/cmd" -m "PING"

C:\Program Files\mosquitto\mosquitto_pub.exe -h test.mosquitto.org -t "tfm_clara_demo/cmd" -m "LED_ON"

C:\Program Files\mosquitto\mosquitto_pub.exe -h test.mosquitto.org -t "tfm_clara_demo/cmd" -m "LED_OFF"

En la consola suscrita a todo el *namespace* (tfm_clara_demo/#) se observa la secuencia completa (ver Figura 32), en la que el comando enviado en cmd y la confirmación inmediata en ack, con *payloads* von el siguiente formato:

{"ack":"PONG","ms":<timestamp>}, {"ack":"LED_ON"} y {"ack":"LED_OFF"}.

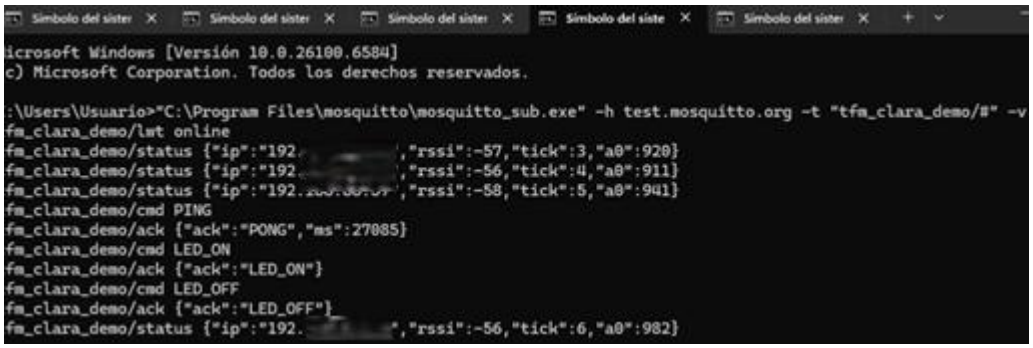


Figura 32: Secuencia de comandos y respuestas en la consola suscrita (tfm_clara_demo/#). Fuente: elaboración propia.

Finalmente, se mide la latencia de ida y vuelta (Round Trip Time, RTT) para cuantificar el tiempo entre el envío de un comando PING y la recepción del PONG correspondiente. El proceso se automatiza mediante un script en PowerShell (ver Anexo D) que envía diez comandos y calcula los tiempos individuales y agregados.

El script registra los RTTs y calcula automáticamente el valor mínimo, promedio y máximo. Se observan los siguientes resultados (Figura 33):

- Mínimo: **134 ms**
- Promedio: **334,5 ms**
- Máximo: **1890,7 ms**

```
{"ack":"PONG","ms":4576777}
{"ack":"PONG","ms":4577131}
{"ack":"PONG","ms":4577520}
PS C:\Users\Usuario>
PS C:\Users\Usuario> "RTTs: $results"
RTTs: 1890.7 183 222.6 142.9 149.1 140.2 134 145.5 163.4 173.7
PS C:\Users\Usuario> $min = ($results | Measure-Object -Minimum).Minimum
PS C:\Users\Usuario> $avg = [math]::Round(($results | Measure-Object -Average).Average, 1)
PS C:\Users\Usuario> $max = ($results | Measure-Object -Maximum).Maximum
PS C:\Users\Usuario> "min/avg/max = $min / $avg / $max ms"
min/avg/max = 134 / 334.5 / 1890.7 ms
PS C:\Users\Usuario>
```

Figura 33: Medición de latencia RTT con el script de PowerShell. Fuente: elaboración propia.

Los valores medidos evidencian la latencia real del sistema cuando opera sobre un *broker* público. Aunque la variabilidad es notable, las cifras se encuentran en un rango aceptable para aplicaciones IoT de monitorización no crítico, en las que una respuesta del orden de cientos de milisegundos es aceptable.

Para ofrecer una visión global del proceso desarrollado, la Figura 34 resume de forma esquemática las etapas de implementación y validación realizadas del sistema IoT. En la parte superior se representan las fases de desarrollo, desde la conexión Wi-Fi inicial hasta el cliente MQTT extendido con jerarquía de *topics*, y en la parte inferior se muestran las pruebas de validación realizadas sobre el sistema, que abarcan la verificación del estado LWT, la telemetría periódica, la ejecución de comandos y la medición de latencia.

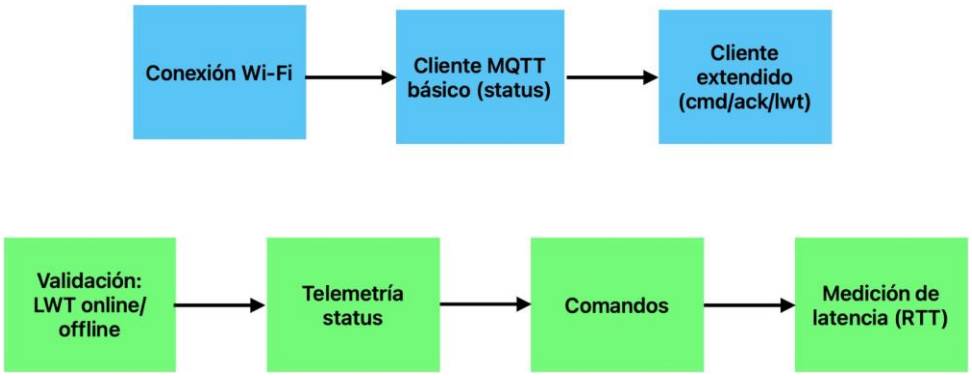


Figura 34: Esquema general del proceso de desarrollo y validación del sistema IoT; las etapas en azul corresponden al desarrollo y las verdes a la validación. Fuente: elaboración propia.

6.2 Ingesta y procesamiento en tiempo real con Spark y PostgreSQL

Este apartado aborda el pipeline de ingesta y procesamiento en tiempo real, en el cual se reciben, transforman y almacenan los datos procedentes de la capa IoT. Esta fase es el núcleo funcional del sistema, ya que conecta el dispositivo físico basado en la placa Seeed XIAO ESP32-C3 con la infraestructura analítica y de almacenamiento, y garantiza que las mediciones publicadas a través del protocolo MQTT se procesen de forma continua y trazable.

Para ello, se emplea Apache Spark Structured Streaming, una tecnología orientada al procesamiento incremental de datos en micro-lotes, la cual permite aplicar transformaciones, deduplicar registros y mantener la consistencia incluso ante interrupciones o variaciones en la

tasa de llegada. Aunque el volumen de datos generado por el dispositivo es reducido, se opta por su uso con un propósito demostrativo y de escalabilidad futura, de modo que la infraestructura pueda adaptarse fácilmente a entornos con múltiples dispositivos o canales de monitorización en paralelo.

Los datos resultantes se almacenan en dos niveles:

- **PostgreSQL**, el cual actúa como base transaccional para las consultas operativas y la verificación inmediata de las mediciones.
- **Un repositorio analítico en formato Parquet**, que funciona como una base tipo *Data Lake local* y que tiene como objetivo conservar el histórico de datos completo y facilitar análisis avanzados o el entrenamiento de modelos de ML en etapas futuras de investigación.

Dado que Spark no puede consumir directamente desde un broker MQTT, se implementa un bridge intermedio que serializa los mensajes en formato NDJSON (newline-delimited JSON). Este componente actúa como buffer de persistencia temporal y asegura que ningún evento se pierda aunque el motor de procesamiento se detenga.

A partir de esta infraestructura base, el sistema evoluciona hacia un procesamiento más avanzado que incorpora dos funcionalidades importantes:

- Por un lado, el cálculo de agregados temporales (por ventanas de 1 y 5 minutos) que permiten analizar la evolución de las métricas clínicas con distintos niveles de granularidad.
- Y por otro, la generación de alertas en tiempo real, basada en reglas simples de umbrales que detectan desviaciones de calidad en las mediciones. Estas alertas proporcionan explicabilidad al comportamiento del sistema, ya que permiten vincular los fallos o incidencias detectados por el modelo de ML con causas técnicas concretas.

Esta etapa transforma los eventos brutos generados por la capa IoT en información estructurada, procesada y accionable que está para su visualización en paneles de control. Con ello, se completa la integración extremo a extremo entre la captura IoT, la ingesta en tiempo real y el análisis de los datos, consolidando la base técnica del sistema. La Figura 35 muestra el flujo general de este proceso, que integra la publicación de datos desde el dispositivo IoT, la ingesta y el procesamiento en tiempo real mediante Spark Structured Streaming y la persistencia final en bases de datos transaccionales y analíticas.

Figura 35: Flujo de ingesta, procesamiento y almacenamiento en tiempo real mediante MQTT, Spark Structured Streaming y PostgreSQL. Fuente: elaboración propia.

6.2.1 Configuración de la capa de persistencia transaccional con PostgreSQL en Docker

Como primer paso en la fase de ingesta y procesamiento en tiempo real, se configura la capa de persistencia transaccional mediante el despliegue de una instancia de PostgreSQL en contenedor. Una vez iniciado el servicio y abierto el puerto 5432, se verifica su correcto funcionamiento desde la interfaz de *Docker Desktop* (véase Figura 36). A continuación, se accede a la base de datos *tfm* mediante el cliente DBeaver, donde se llevan a cabo las operaciones de creación y comprobación del esquema.

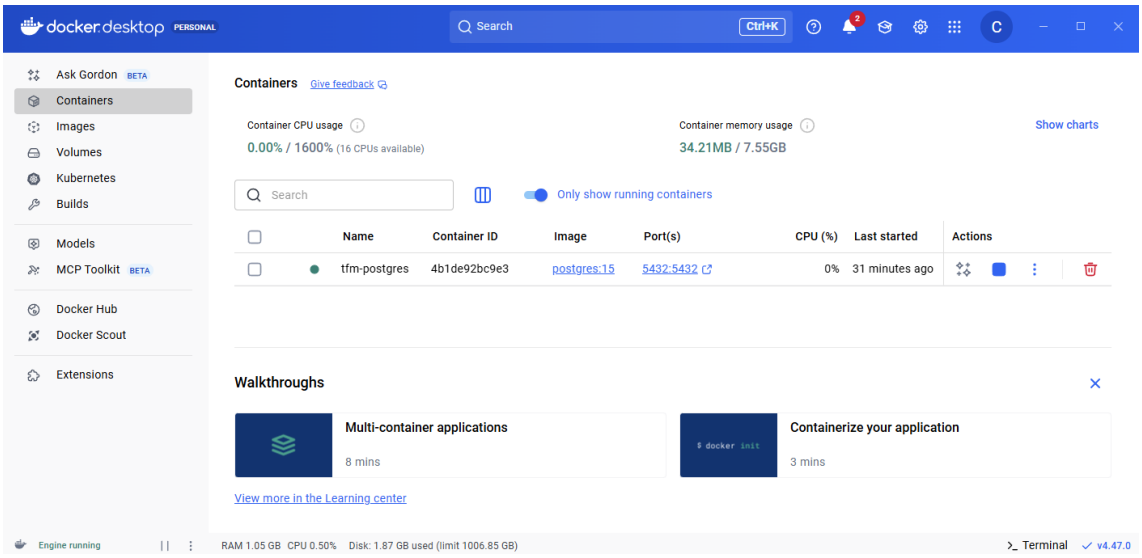


Figura 36: Contenedor de PostgreSQL desplegado en Docker Desktop en estado running. Fuente: elaboración propia.

En este contexto, se define la tabla trc_observation para almacenar cada observación medida por el dispositivo y procesada posteriormente en la capa de streaming. El código empleado para su definición se incluye en el Anexo E. Después de crearla, se valida su disponibilidad en el árbol de objetos del esquema public (véase Figura37).

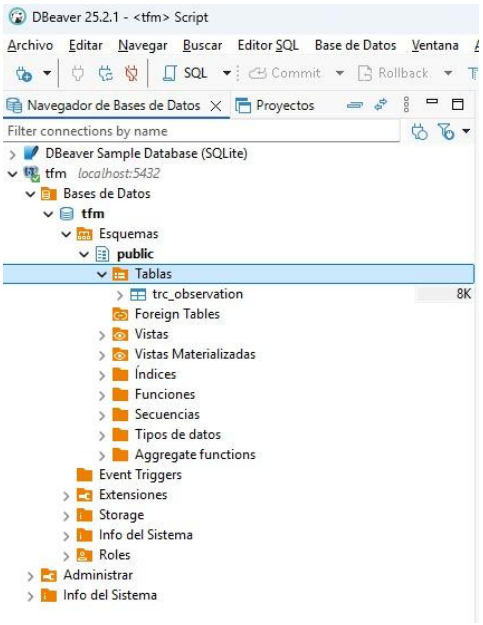


Figura 37: Tabla trc_observation creada en el esquema público de la base de datos PostgreSQL. Fuente: elaboración propia.

La tabla trc_observation está compuesta por los siguientes campos:

- **ts**: registra la marca temporal de la observación y permite ordenar los registros, agruparlos por ventanas temporales y sincronizarlos con otras fuentes.
- **device_id**: identifica el dispositivo que genera la medición y, junto con ts, forma la clave mínima para la integridad de cada registro.
- **patient_id**: contiene un identificador anonimizado que posibilita seguir la evolución de un mismo sujeto sin comprometer su privacidad.
- **rssi**: refleja la intensidad de la señal en el momento de la transmisión, lo cual resulta útil para evaluar la calidad de la comunicación.
- **trc_ms**: almacena el TRC en milisegundos.
- **qa** *(quality assurance)*: contiene indicadores de calidad de la medición (como saturación, artefactos o presión aplicada) que pueden evolucionar con nuevas versiones del firmware.
- **env**: almacena información contextual (temperatura, iluminación, etc.) útil para interpretar variaciones en las mediciones.
- **raw_payload**: conserva el mensaje completo recibido en formato JSON para garantizar la trazabilidad de los datos.

- **ingested_at**: se completa automáticamente en el momento de la inserción y permite medir la latencia extremo a extremo, además de reconstruir el orden real de llegada de los mensajes.

Con esta estructura, la base de datos queda preparada para recibir inserciones en tiempo real desde la capa de procesamiento y soportar consultas inmediatas, como la obtención de las últimas observaciones por dispositivo o paciente, al mismo tiempo que conserva el payload para la trazabilidad y análisis.

6.2.2 Validación inicial de la ingesta de datos con Spark en modo batch

Antes de llevar a cabo el procesamiento en tiempo real de los datos, se realiza una fase preliminar en modo *batch* con Apache Spark, la cual tiene como objetivo validar la capacidad del sistema para leer datos en formato JSON, transformarlos y almacenarlos en la BBDD PostgreSQL. Este paso intermedio es fundamental para comprobar que la integración entre los distintos componentes de la arquitectura es correcta antes de realizar la implementación con Spark Structured Streaming.

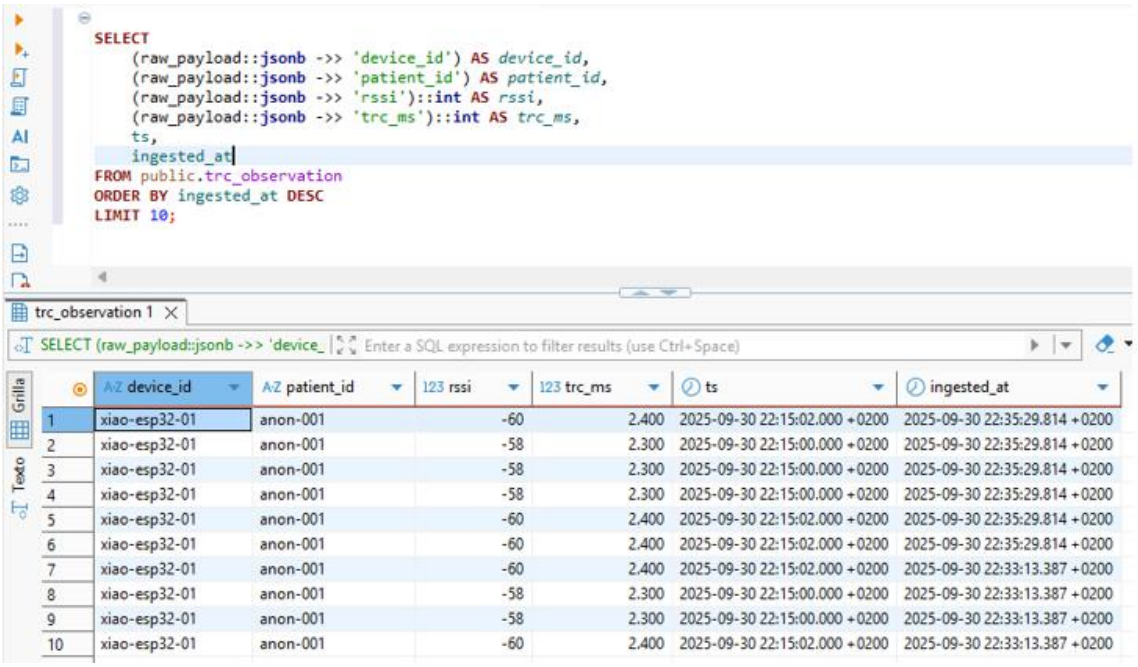
En esta prueba se desarrolla un *job* en PySpark llamado ingest_files_once.py y ejecutado mediante spark-submit (ver Anexos F y G). El flujo seguido consiste en la lectura de ficheros JSON guardados en la carpeta *incoming*, la aplicación de transformaciones básicas, como la conversión de tipos, la normalización de campos y la creación del campo raw_payload que almacena el JSON completo para trazabilidad, y la posterior inserción en la tabla trc_observation de PostgreSQL.

Este diseño permite verificar tanto la conectividad con la base de datos como la consistencia de los datos almacenados (véase Figura 38).

Figura 38: Flujo de validación de ingesta en modo batch con Spark y PostgreSQL. Fuente: elaboración propia.

La validación se realiza ejecutando el *job* sobre diferentes lotes de prueba (lote1.json, lote2.json, lote3.json). Posteriormente, los resultados se analizan en DBeaver (Figura 39), confirmando que las filas se insertan correctamente y que los tipos de datos coinciden con lo esperado, siendo, por ejemplo, enteros para campos métricos y jsonb para los campos anidados.

La inclusión del campo raw_payload muestra ser especialmente útil, ya que permite consultar el JSON original incluso en caso de fallos en las transformaciones (ver Anexo H).



```
SELECT
    (raw_payload::jsonb ->> 'device_id') AS device_id,
    (raw_payload::jsonb ->> 'patient_id') AS patient_id,
    (raw_payload::jsonb ->> 'rssi')::int AS rssi,
    (raw_payload::jsonb ->> 'trc_ms')::int AS trc_ms,
    ts,
    ingested_at
FROM public.trc_observation
ORDER BY ingested_at DESC
LIMIT 10;
```

trc_observation 1

SELECT (raw_payload::jsonb ->> 'device_ | Enter a SQL expression to filter results (use Ctrl+Space)

	device_id	patient_id	rssi	trc_ms	ts	ingested_at
1	xiao-esp32-01	anon-001	-60	2.400	2025-09-30 22:15:02.000 +0200	2025-09-30 22:35:29.814 +0200
2	xiao-esp32-01	anon-001	-58	2.300	2025-09-30 22:15:00.000 +0200	2025-09-30 22:35:29.814 +0200
3	xiao-esp32-01	anon-001	-58	2.300	2025-09-30 22:15:00.000 +0200	2025-09-30 22:35:29.814 +0200
4	xiao-esp32-01	anon-001	-58	2.300	2025-09-30 22:15:00.000 +0200	2025-09-30 22:35:29.814 +0200
5	xiao-esp32-01	anon-001	-60	2.400	2025-09-30 22:15:02.000 +0200	2025-09-30 22:35:29.814 +0200
6	xiao-esp32-01	anon-001	-60	2.400	2025-09-30 22:15:02.000 +0200	2025-09-30 22:35:29.814 +0200
7	xiao-esp32-01	anon-001	-60	2.400	2025-09-30 22:15:02.000 +0200	2025-09-30 22:33:13.387 +0200
8	xiao-esp32-01	anon-001	-58	2.300	2025-09-30 22:15:00.000 +0200	2025-09-30 22:33:13.387 +0200
9	xiao-esp32-01	anon-001	-58	2.300	2025-09-30 22:15:00.000 +0200	2025-09-30 22:33:13.387 +0200
10	xiao-esp32-01	anon-001	-60	2.400	2025-09-30 22:15:02.000 +0200	2025-09-30 22:33:13.387 +0200

Figura 39: Consulta en DBeaver mostrando datos insertados y campos extraídos desde raw_payload. Fuente: elaboración propia.

6.2.3 Validación de la ingesta en tiempo real con Spark Structured Streaming (datos simulados)

Tras la fase preliminar en *batch*, se diseña una prueba en modo *streaming* con Spark Structured Streaming. El objetivo es simular la llegada continua de datos en formato JSON y validar que la arquitectura es capaz de procesarlos en micro-lotes, aplicar transformaciones básicas y escribir los resultados de forma incremental en la base de datos PostgreSQL. (49)

Para simular el flujo de datos, se utiliza un script en Python que crea ficheros JSON de prueba (Anexo I). Estos ficheros contienen registros similares a los producidos por el dispositivo real con la *xiao-esp32-01*, con campos de telemetría (rssi, trc_ms) y parámetros anidados (qa, env). Cada ejecución del script crea un nuevo archivo en la carpeta *incoming_streaming* y consigue simular de esta forma la llegada de datos desde el entorno IoT.

La ingesta se implementa mediante un *job* en PySpark llamado ingest_streaming.py (Anexo J). Este *job* define el esquema de los datos, lee en modo *streaming* la carpeta configurada, transforma las columnas necesarias (conversión de ts a tipo *timestamp*, creación del campo raw_payload para trazabilidad, añadir de la marca temporal ingested_at) y aplica un mecanismo de deduplicación por micro-lote.

La ejecución se realiza con el comando spark-submit mostrado en el Anexo K, en el que se especifica tanto la ruta al intérprete de Python como la dependencia del driver PostgreSQL. Una vez iniciado, el proceso permanece en ejecución continua a la espera de nuevos ficheros en la carpeta.

La validación de la ingesta es de dos fases:

1. **Inserción de datos simulados.** Se ejecuta el generador de JSON en varias ocasiones, comprobando que en cada ejecución se añaden nuevos registros en la tabla trc_observation. Las Figuras 40 y 41 muestran el resultado de estas pruebas, en las que se puede apreciar que tras la primera carga se insertan únicamente dos filas, mientras que en ejecuciones posteriores se acumulan nuevos registros.
2. **Consultas de verificación en PostgreSQL.** Mediante DBeaver se ejecutan distintas consultas SQL para confirmar la integridad del proceso. Se utiliza una consulta básica (Anexo L) para comprobar el número total de filas y extraer campos del JSON almacenado en raw_payload. Además, se diseñan consultas avanzadas (Anexo M) para detectar posibles duplicados, calcular *hashes* de los mensajes y estimar la latencia entre el momento de inserción y la consulta.

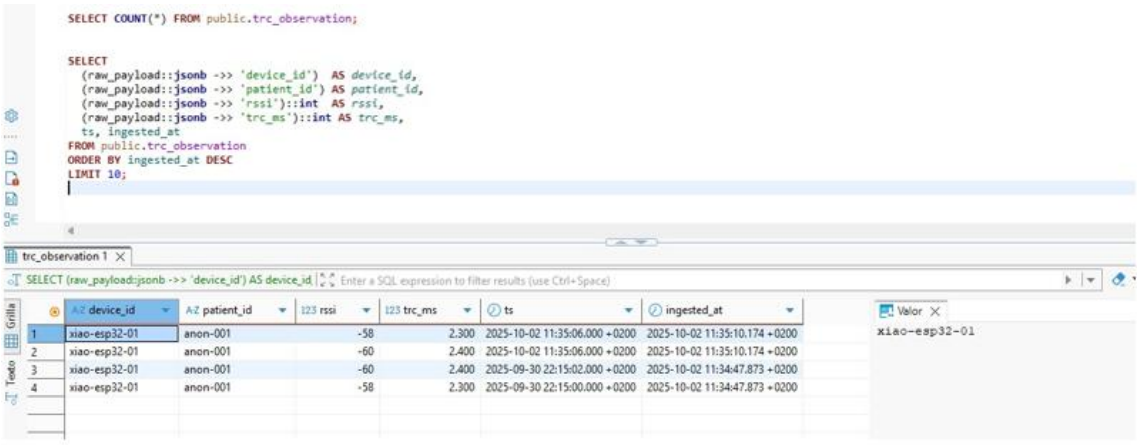


Figura 40: Verificación en DBeaver tras la primera carga de ficheros JSON en modo streaming. Fuente: elaboración propia.

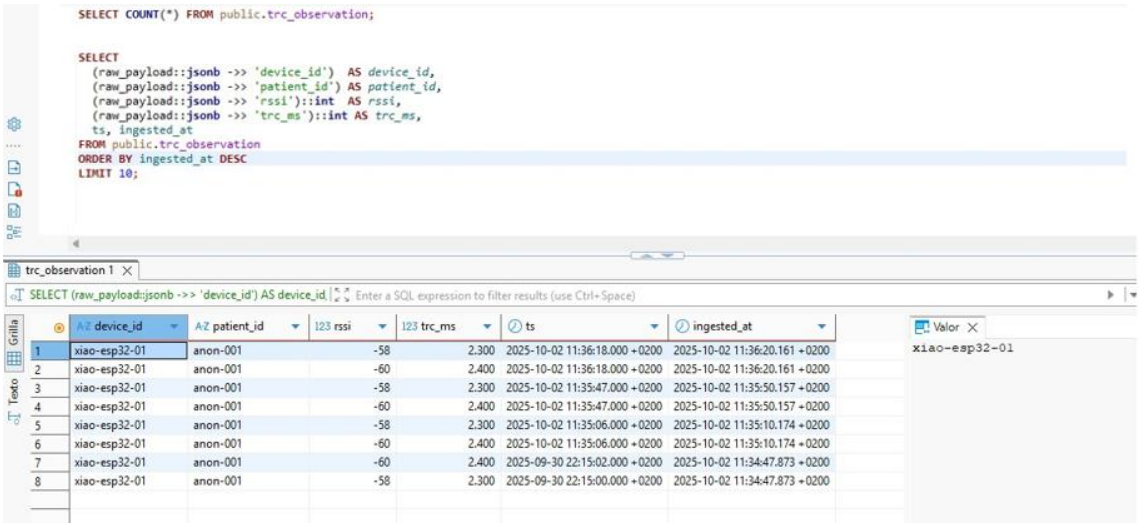


Figura 41: Acumulación de registros en la tabla trc_observation tras varias ejecuciones del generador de datos. Fuente: elaboración propia.

Los resultados confirman que el sistema procesa en tiempo real los ficheros generados, los transforma y los almacena correctamente en PostgreSQL. Las verificaciones adicionales de duplicados y latencia demuestran que, aunque el pipeline todavía es simple, ya proporciona calidad y robustez de los datos.

6.2.4 Ingesta de datos IoT en tiempo real mediante MQTT y Spark Structured Streaming

Tras validar la arquitectura de streaming con datos simulados, se realiza ahora la integración real entre la capa IoT y el pipeline de procesamiento en tiempo real.

Para ello, se adapta el firmware de la placa Seeed XIAO ESP32-C3 (Anexo N) para publicar periódicamente mensajes en formato JSON, siguiendo un esquema que incluye los campos principales de telemetría (v, ts, device_id, patient_id, rssi, trc_ms), los parámetros anidados (qa, env) y un bloque extensible data con información adicional como la dirección IP, el contador de ticks y una lectura analógica.

El dispositivo también implementa la suscripción a comandos entrantes (tfm_clara_demo/cmd), con confirmaciones en ack, y gestiona de forma robusta su estado de conexión mediante el mecanismo Last Will and Testament (LWT) en el topic tfm_clara_demo/lwt (ver Figura 42).

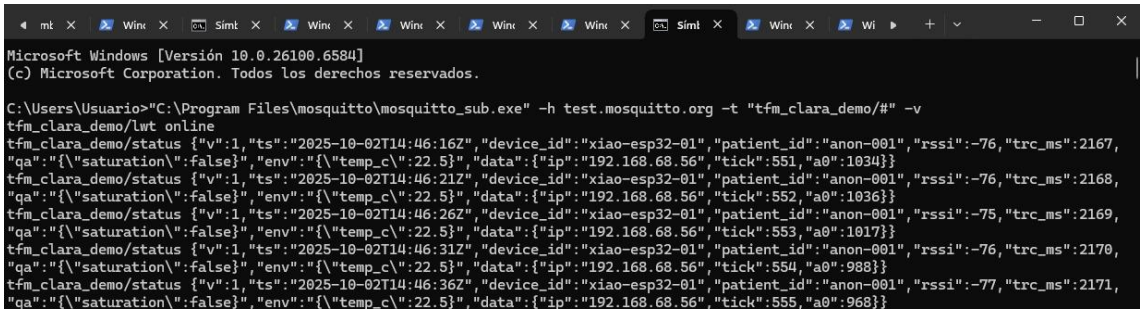


Figura 42: Visualización en consola con mosquitto_sub de los topics tfm_clara_demo/# mostrando estado (LWT) y telemetría. Fuente: elaboración propia.

Como Spark Structured Streaming no puede conectarse directamente a un broker MQTT, se desarrolla un bridge en Python (Anexo O) que actúa como intermediario. Este script se suscribe al topic de telemetría y escribe los mensajes en formato NDJSON dentro de la carpeta incoming_streaming (Figura 43), rotando los archivos cada 15 segundos para facilitar su ingestión incremental. Su ejecución se muestra en la Figura 44.

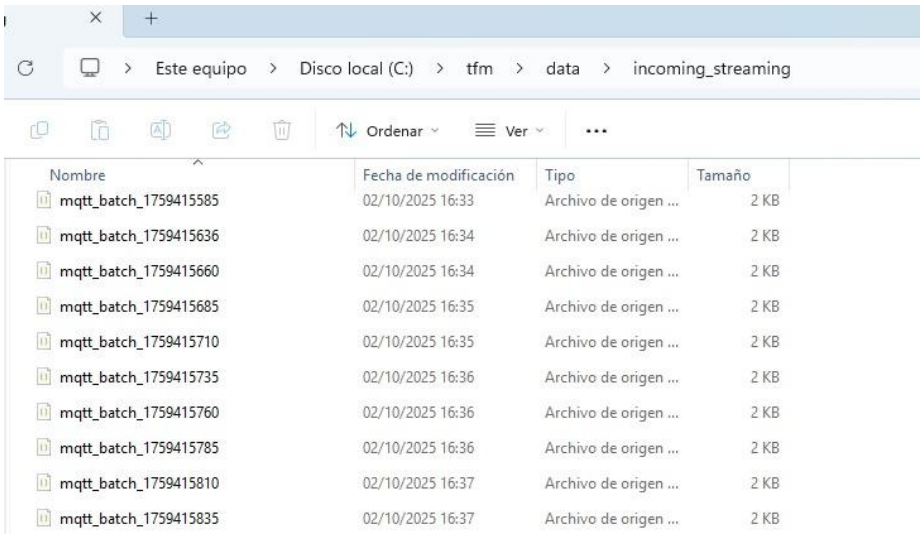


Figura 43: Carpeta incoming_streaming con los lotes JSON generados por el bridge MQTT. Fuente: elaboración propia.

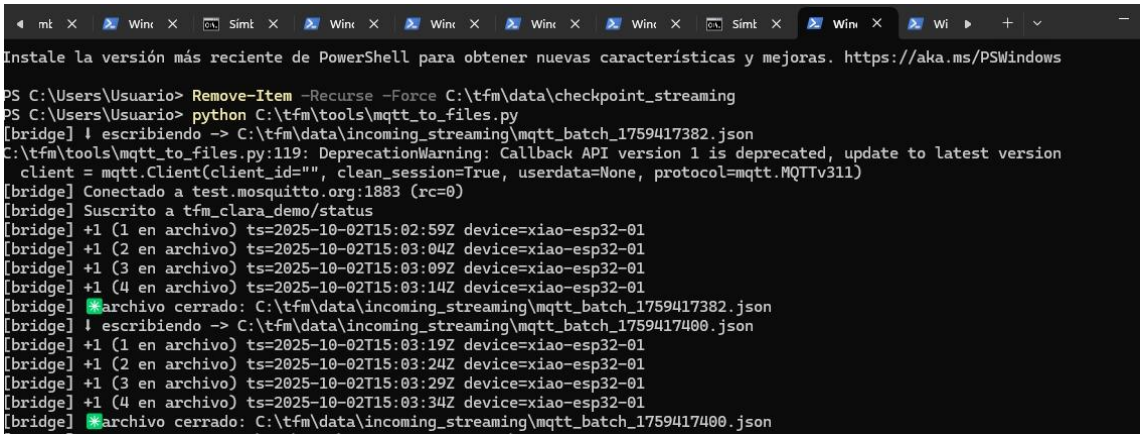


Figura 44: Ejecución del bridge mqtt_to_files.py en PowerShell, con rotación periódica de archivos NDJSON. Fuente: elaboración propia.

La siguiente ingesta se lleva a cabo mediante un job en PySpark (Anexo J) que se ejecuta con spark-submit (Anexo K), el cual monitoriza en tiempo real los ficheros generados por el bridge. Este proceso aplica el esquema definido, convierte ts a formato nativo, añade raw_payload para conseguir trazabilidad, la columna ingested_at como marca de inserción y elimina duplicados dentro de cada micro-lote.

Cada batch se escribe en PostgreSQL dentro de la tabla trc_observation (Anexo E), integrando así los datos IoT en la capa de almacenamiento persistente. La Figura 44 muestra una consulta realizada en DBeaver, donde se observan las columnas nativas y los campos extraídos del objeto JSON tras el proceso de ingesta.

```
SELECT
  (raw_payload::jsonb ->> 'device_id') AS device_id,
  (raw_payload::jsonb ->> 'patient_id') AS patient_id,
  (raw_payload::jsonb ->> 'rssi')::int  AS rssi,
  (raw_payload::jsonb ->> 'trc_ms')::int AS trc_ms,
  ts, ingested_at
FROM public.trc_observation
ORDER BY ingested_at DESC
LIMIT 10;
```

trc_observation 1

SELECT (raw_payload::jsonb ->> 'device_id') AS device_id, | Enter a SQL expression to filter results (use Ctrl+Space)

	device_id	patient_id	rssi	trc_ms	ts	ingested_at
1	xiao-esp32-01	anon-001	-73	2.704	02 17:02:26.000	0-02 17:02:30.166 +0200
2	xiao-esp32-01	anon-001	-75	2.700	02 17:02:06.000	0-02 17:02:10.180 +0200
3	xiao-esp32-01	anon-001	-73	2.697	02 17:01:51.000	0-02 17:01:55.171 +0200
4	xiao-esp32-01	anon-001	-74	2.693	02 17:01:31.000	0-02 17:01:35.175 +0200
5	xiao-esp32-01	anon-001	-74	2.690	02 17:01:16.000	0-02 17:01:20.182 +0200
6	xiao-esp32-01	anon-001	-75	2.687	02 17:01:01.000	0-02 17:01:05.169 +0200
7	xiao-esp32-01	anon-001	-74	2.683	02 17:00:41.000	0-02 17:00:45.173 +0200
8	xiao-esp32-01	anon-001	-75	1.753	02 17:00:25.000	0-02 17:00:30.178 +0200
9	xiao-esp32-01	anon-001	-73	1.749	02 17:00:05.000	0-02 17:00:10.172 +0200
10	xiao-esp32-01	anon-001	-76	1.640	02 16:59:40.000	0-02 16:59:45.185 +0200

Figura 44: Consulta en DBeaver mostrando las columnas nativas y los campos extraídos del JSON. Fuente: elaboración propia.

Finalmente, la latencia se evalúa mediante una consulta SQL (Anexo T) que calcula la diferencia entre la hora actual y la última marca ingested_at. El resultado (Figura 45) muestra una diferencia de unos 17 segundos, por lo que confirma que el sistema procesa los mensajes en tiempo casi real, teniendo en cuenta el retardo natural del bridge y del micro-batch de Spark.

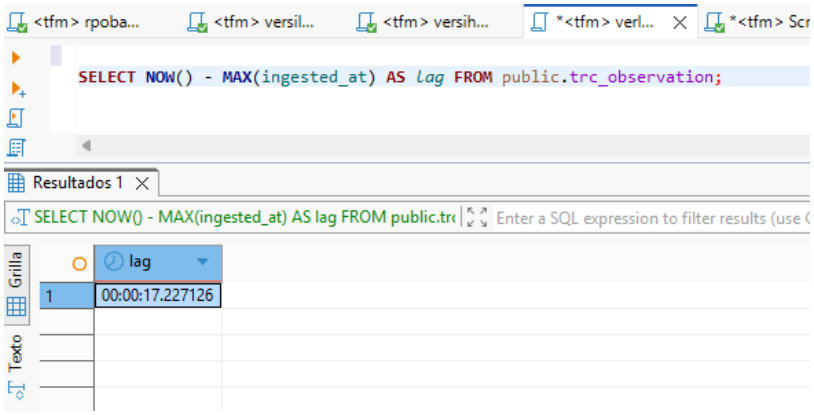


Figura 45: Medición de latencia aproximada entre publicación e ingesta con consulta SQL en PostgreSQL. Fuente: elaboración propia.

6.2.5 Publicación y procesamiento de métricas clínicas (CRT, SQI, fuerza, luz ambiente)

En esta fase se amplía la ingesta de datos IoT para incluir además de la telemetría básica previamente desarrollada, las métricas clínicas relevantes para el caso de uso.

El dispositivo con la ESP32-C3publica eventos en formato JSON que distinguen dos tipos de mensajes través del firmware desarrollado (Anexo P):

- **status**: indican el latido periódico del dispositivo e incluyen la información básica, como el identificador, el nivel de señal RSSI y la dirección IP.
- **crt**: corresponden con los eventos clínicos que recogen las métricas del TRC junto con los parámetros asociados de calidad.

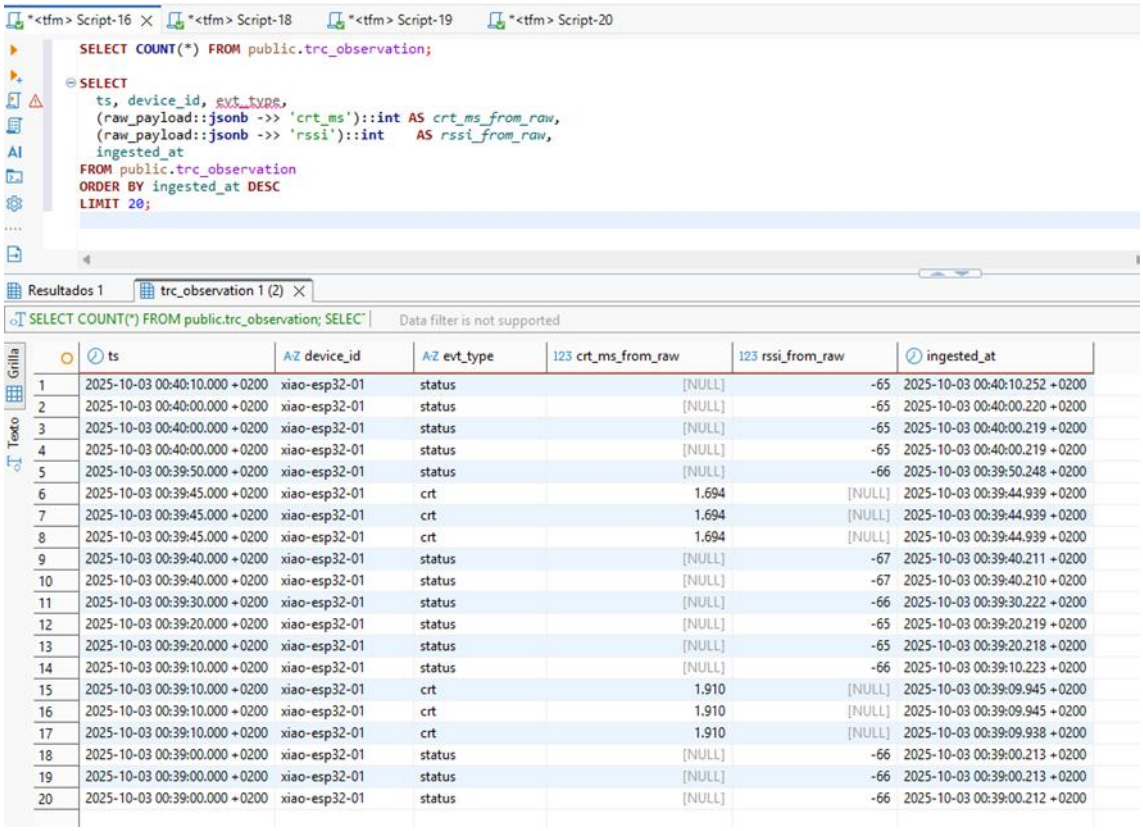
Los mensajes crt incluyen tanto valores temporales (t_compress_start, t_compress_end, t_recovery_end, crt_ms) como variables ópticas (color_baseline, color_min, color_recovery_pct), de iluminación (ambient_lux), de fuerza aplicada (force_peak_n, force_hold_ms) y un índice de calidad de señal (sqi). También se añade un campo opcional trace con trazas reducidas de fuerza y color para poder reconstruir la forma de la curva sin tener que mandar las muestra completas.

Para evitar dependencias directas entre el dispositivo y Spark, se mantiene un bridge entre MQTT y NDJSON (Anexo Q) que se suscribe a los topics del namespace y muestra cada mensaje recibido como una línea independiente en disco, lo que garantiza la persistencia de los eventos incluso si Spark no está activo.

Spark Structured Streaming (Anexo R) procesa los ficheros en micro-lotes, normaliza los campos principales y escribe los resultados en la tabla public.trc_observation de PostgreSQL (definida en el Anexo E y ampliada en el Anexo S), además de generar un histórico en formato Parquet. La columna raw_payload conserva el JSON íntegro de cada mensaje, lo que permite acceder a métricas adicionales sin tener que modificar el esquema transaccional.

La verificación de la ingesta se realiza en DBeaver (Anexo T).

La Figura 46 muestra los mensajes de tipo status y crt almacenados en la tabla trc_observation, mientras que la Figura 47 presenta los valores clínicos extraídos de los eventos crt, confirmando la persistencia de las métricas junto con los metadatos técnicos de cada evento (ts, device_id, ingested_at).



```
SELECT COUNT(*) FROM public.trc_observation;

SELECT
  ts, device_id, evt_type,
  (raw_payload::jsonb ->> 'crt_ms')::int AS crt_ms_from_raw,
  (raw_payload::jsonb ->> 'rssi')::int    AS rssi_from_raw,
  ingested_at
FROM public.trc_observation
ORDER BY ingested_at DESC
LIMIT 20;
```

	ts	device_id	evt_type	crt_ms_from_raw	rssi_from_raw	ingested_at
1	2025-10-03 00:40:10.000 +0200	xiao-esp32-01	status	[NULL]	-65	2025-10-03 00:40:10.252 +0200
2	2025-10-03 00:40:00.000 +0200	xiao-esp32-01	status	[NULL]	-65	2025-10-03 00:40:00.220 +0200
3	2025-10-03 00:40:00.000 +0200	xiao-esp32-01	status	[NULL]	-65	2025-10-03 00:40:00.219 +0200
4	2025-10-03 00:40:00.000 +0200	xiao-esp32-01	status	[NULL]	-65	2025-10-03 00:40:00.219 +0200
5	2025-10-03 00:39:50.000 +0200	xiao-esp32-01	status	[NULL]	-66	2025-10-03 00:39:50.248 +0200
6	2025-10-03 00:39:45.000 +0200	xiao-esp32-01	crt	1.694	[NULL]	2025-10-03 00:39:44.939 +0200
7	2025-10-03 00:39:45.000 +0200	xiao-esp32-01	crt	1.694	[NULL]	2025-10-03 00:39:44.939 +0200
8	2025-10-03 00:39:45.000 +0200	xiao-esp32-01	crt	1.694	[NULL]	2025-10-03 00:39:44.939 +0200
9	2025-10-03 00:39:40.000 +0200	xiao-esp32-01	status	[NULL]	-67	2025-10-03 00:39:40.211 +0200
10	2025-10-03 00:39:40.000 +0200	xiao-esp32-01	status	[NULL]	-67	2025-10-03 00:39:40.210 +0200
11	2025-10-03 00:39:30.000 +0200	xiao-esp32-01	status	[NULL]	-66	2025-10-03 00:39:30.222 +0200
12	2025-10-03 00:39:20.000 +0200	xiao-esp32-01	status	[NULL]	-65	2025-10-03 00:39:20.219 +0200
13	2025-10-03 00:39:20.000 +0200	xiao-esp32-01	status	[NULL]	-65	2025-10-03 00:39:20.218 +0200
14	2025-10-03 00:39:10.000 +0200	xiao-esp32-01	status	[NULL]	-66	2025-10-03 00:39:10.223 +0200
15	2025-10-03 00:39:10.000 +0200	xiao-esp32-01	crt	1.910	[NULL]	2025-10-03 00:39:09.945 +0200
16	2025-10-03 00:39:10.000 +0200	xiao-esp32-01	crt	1.910	[NULL]	2025-10-03 00:39:09.945 +0200
17	2025-10-03 00:39:10.000 +0200	xiao-esp32-01	crt	1.910	[NULL]	2025-10-03 00:39:09.938 +0200
18	2025-10-03 00:39:00.000 +0200	xiao-esp32-01	status	[NULL]	-66	2025-10-03 00:39:00.213 +0200
19	2025-10-03 00:39:00.000 +0200	xiao-esp32-01	status	[NULL]	-66	2025-10-03 00:39:00.213 +0200
20	2025-10-03 00:39:00.000 +0200	xiao-esp32-01	status	[NULL]	-66	2025-10-03 00:39:00.212 +0200

Figura 46: Consulta en DBeaver con eventos status y crt. Fuente: elaboración propia.

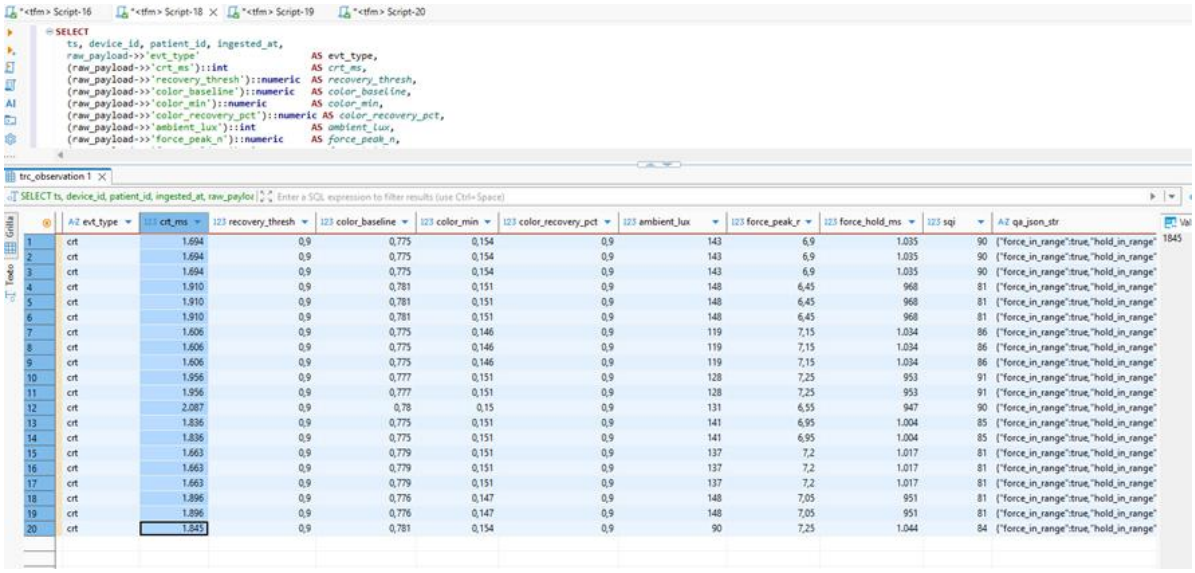


Figura 47: Consulta en DBeaver con métricas clínicas detalladas de los eventos crt. Fuente: elaboración propia.

Este flujo demuestra que el sistema no solo recibe y persiste los datos de telemetría básica, sino que también trabaja con la información clínica estructurada en tiempo real.

A continuación, se detalla el significado de las métricas enviadas:

- crt_ms (Capillary Refill Time, ms): tiempo que tarda el lecho ungueal en recuperar su color tras una compresión. Valores superiores a 2000 ms pueden asociarse a hipoperfusión periférica.
- color_baseline y color_min: referencias ópticas para calcular el CRT (el primero antes de la compresión, el segundo durante el mínimo).
- color_recovery_pct: grado de recuperación respecto al baseline (se usa el 90 % como criterio de fin del CRT).
- ambient_lux: luz ambiental, ya que afectan a la calidad de la señal.
- force_peak_n y force_hold_ms: fuerza máxima y duración de la compresión.
- sqi (Signal Quality Index): índice de calidad (0–100) que pondera fuerza, duración, movimiento y forma de curva.
- qa: con indicadores como force_in_range, hold_in_range y motion_high.
- trace: puntos representativos de fuerza y color para validar la forma de la curva.

6.2.6 Agregación y alertas en tiempo real con Spark Structured Streaming

Después de implementar la ingesta básica en streaming, se añade en esta última etapa de desarrollo una capa de procesamiento para generar información útil tanto para la visualización como para la supervisión de la calidad de los datos.

En primer lugar, se incorporan agregados por ventana temporal (ver Figuras 48 y 49). Además de almacenar los eventos clínicos individuales, el sistema calcula métricas resumidas por

dispositivo y paciente en intervalos de 1 minuto y 5 minutos, lo cual permite detectar rápidamente variaciones (1 min) y obtener una visión más estable y suavizada de la señal (5 min).

	id	window_start	window_end	device_id	patient_id	n_events	crt_ms_mean	crt_ms_p95	sqi_mean	created_at
1	115	2025-10-04 17:12:00.000 +0200	2025-10-04 17:13:00.000 +0200	xiao-esp32-01	anon-001	1	1.706	1.706	89	2025-10-04 17:16:39.805 +020
2	114	2025-10-04 17:11:00.000 +0200	2025-10-04 17:12:00.000 +0200	xiao-esp32-01	anon-001	1	1.916	1.916	84	2025-10-04 17:14:42.057 +020
3	113	2025-10-04 17:10:00.000 +0200	2025-10-04 17:11:00.000 +0200	xiao-esp32-01	anon-001	1	1.691	1.691	62	2025-10-04 17:14:40.501 +020
4	112	2025-10-04 17:09:00.000 +0200	2025-10-04 17:10:00.000 +0200	xiao-esp32-01	anon-001	1	1.585	1.585	75	2025-10-04 17:13:30.298 +020
5	111	2025-10-04 17:07:00.000 +0200	2025-10-04 17:08:00.000 +0200	xiao-esp32-01	anon-001	1	2.521	2.521	81	2025-10-04 17:11:30.092 +020
6	110	2025-10-04 17:06:00.000 +0200	2025-10-04 17:07:00.000 +0200	xiao-esp32-01	anon-001	1	2.136	2.136	63	2025-10-04 17:10:40.428 +020
7	109	2025-10-04 17:05:00.000 +0200	2025-10-04 17:06:00.000 +0200	xiao-esp32-01	anon-001	1	1.427	1.427	71	2025-10-04 17:10:38.488 +020
8	108	2025-10-04 17:04:00.000 +0200	2025-10-04 17:05:00.000 +0200	xiao-esp32-01	anon-001	1	2.537	2.537	63	2025-10-04 17:08:45.584 +020
9	107	2025-10-04 17:02:00.000 +0200	2025-10-04 17:03:00.000 +0200	xiao-esp32-01	anon-001	1	2.359	2.359	88	2025-10-04 17:06:25.559 +020
10	105	2025-10-04 17:01:00.000 +0200	2025-10-04 17:02:00.000 +0200	xiao-esp32-01	anon-001	1	1.415	1.415	66	2025-10-04 17:05:15.293 +020
11	106	2025-10-04 17:00:00.000 +0200	2025-10-04 17:01:00.000 +0200	xiao-esp32-01	anon-001	1	1.516	1.516	85	2025-10-04 17:05:18.751 +020
12	104	2025-10-04 16:59:00.000 +0200	2025-10-04 17:00:00.000 +0200	xiao-esp32-01	anon-001	1	2.299	2.299	83	2025-10-04 17:03:25.072 +020
13	103	2025-10-04 16:58:00.000 +0200	2025-10-04 16:59:00.000 +0200	xiao-esp32-01	anon-001	1	1.679	1.679	60	2025-10-04 17:02:22.339 +020
14	102	2025-10-04 16:57:00.000 +0200	2025-10-04 16:58:00.000 +0200	xiao-esp32-01	anon-001	1	2.470	2.470	85	2025-10-04 17:01:16.763 +020
15	101	2025-10-04 16:56:00.000 +0200	2025-10-04 16:57:00.000 +0200	xiao-esp32-01	anon-001	1	2.454	2.454	92	2025-10-04 17:00:16.652 +020

Figura 48: Agregados por ventana de 1 minuto en la tabla trc_observation_minute. Fuente: elaboración propia.

SELECT * FROM public.trc_observation_5min ORDER BY window_s

	id	window_start	window_end	device_id	patient_id	n_events	crt_ms_mean	crt_ms_p95	sqi_mean	created_at
1	24	2025-10-04 17:05:00.000 +0200	2025-10-04 17:10:00.000 +0200	xiao-esp32-01	anon-001	4	1.917,25	2.521	72,5	2025-10-04 17:13:36.671 +020
2	23	2025-10-04 17:00:00.000 +0200	2025-10-04 17:05:00.000 +0200	xiao-esp32-01	anon-001	4	1.956,75	2.537	75,5	2025-10-04 17:08:32.965 +020
3	22	2025-10-04 16:55:00.000 +0200	2025-10-04 17:00:00.000 +0200	xiao-esp32-01	anon-001	4	2.225,5	2.470	80	2025-10-04 17:03:16.533 +020
4	21	2025-10-04 16:50:00.000 +0200	2025-10-04 16:55:00.000 +0200	xiao-esp32-01	anon-001	3	1.539,6666666667	1.600	90,3333333333	2025-10-04 16:58:26.368 +020
5	20	2025-10-04 16:45:00.000 +0200	2025-10-04 16:50:00.000 +0200	xiao-esp32-01	anon-001	2	2.232	2.318	73,5	2025-10-04 16:54:19.357 +020
6	19	2025-10-04 16:40:00.000 +0200	2025-10-04 16:45:00.000 +0200	xiao-esp32-01	anon-001	5	1.860,8	1.991	77,6	2025-10-04 16:48:19.886 +020
7	18	2025-10-04 16:35:00.000 +0200	2025-10-04 16:40:00.000 +0200	xiao-esp32-01	anon-001	3	1.878	2.519	89,6666666667	2025-10-04 16:42:43.371 +020
8	2	2025-10-04 16:30:00.000 +0200	2025-10-04 16:35:00.000 +0200	xiao-esp32-01	anon-001	3	1.801,3333333333	2.470	74,6666666667	2025-10-04 16:38:36.866 +020
9	15	2025-10-04 16:20:00.000 +0200	2025-10-04 16:25:00.000 +0200	xiao-esp32-01	anon-001	4	1.956,25	2.503	87	2025-10-04 16:38:41.377 +020
10	17	2025-10-04 16:15:00.000 +0200	2025-10-04 16:20:00.000 +0200	xiao-esp32-01	anon-001	4	1.979,5	2.385	84,25	2025-10-04 16:38:41.911 +020
11	10	2025-10-04 16:10:00.000 +0200	2025-10-04 16:15:00.000 +0200	xiao-esp32-01	anon-001	4	2.300	2.463	77	2025-10-04 16:38:39.222 +020
12	8	2025-10-04 16:05:00.000 +0200	2025-10-04 16:10:00.000 +0200	xiao-esp32-01	anon-001	4	1.960	2.534	79,75	2025-10-04 16:38:38.473 +020
13	3	2025-10-04 16:00:00.000 +0200	2025-10-04 16:05:00.000 +0200	xiao-esp32-01	anon-001	4	2.435,5	2.545	78,25	2025-10-04 16:38:37.487 +020
14	6	2025-10-04 15:55:00.000 +0200	2025-10-04 16:00:00.000 +0200	xiao-esp32-01	anon-001	5	1.759,2	2.086	76,4	2025-10-04 16:38:38.237 +020
15	11	2025-10-04 15:45:00.000 +0200	2025-10-04 15:50:00.000 +0200	xiao-esp32-01	anon-001	5	1.903	2.265	69,8	2025-10-04 16:38:39.300 +020
16	16	2025-10-03 01:10:00.000 +0200	2025-10-03 01:15:00.000 +0200	xiao-esp32-01	anon-001	23	1.857,3913043478	2.041	85,0434782609	2025-10-04 16:38:41.557 +020

Figura 49: Agregados por ventana de 5 minutos en la tabla trc_observation_5min. Fuente: elaboración propia.

En paralelo, se integra un módulo de generación de alertas en tiempo real. Se definen reglas simples sobre las métricas recibidas, por ejemplo con un TRC superior a 2 s, luz ambiental baja, fuerza fuera de rango o baja calidad de señal. Cada vez que se incumple una condición, se registra un evento de alerta en la base de datos con su contexto mínimo (ver Figura 50).

SELECT ts_event, device_id, rule_id, severity, ctx::jsonb, created_at

	ts_event	device_id	rule_id	severity	ctx	created_at
5	2025-10-04 17:02:41.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2359, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 17:03:02.030 +0200
6	2025-10-04 16:59:41.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2299, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 17:00:00.300 +0200
7	2025-10-04 16:59:55.462	xiao-esp32-01	SQI_LOW	warning	{"sqi": 55, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 16:59:55.462 +0200
8	2025-10-04 16:59:55.462	xiao-esp32-01	FORCE_OUT	warning	{"device_id": "xiao-esp32-01", "patient_id": "anon-001", "force_peak_n": 9.5}	2025-10-04 16:59:55.462 +0200
9	2025-10-04 16:59:55.462	xiao-esp32-01	LIGHT_LOW	warning	{"device_id": "xiao-esp32-01", "patient_id": "anon-001", "ambient_lux": 15}	2025-10-04 16:59:55.462 +0200
10	2025-10-04 16:57:41.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2470, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 16:57:56.201 +0200
11	2025-10-04 16:56:41.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2454, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 16:57:00.236 +0200
12	2025-10-04 16:48:41.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2146, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 16:48:57.266 +0200
13	2025-10-04 16:47:41.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2318, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 16:48:00.715 +0200
14	2025-10-04 16:38:36.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2519, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 16:38:52.978 +0200
15	2025-10-04 16:34:14.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2470, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 16:34:21.652 +0200
16	2025-10-04 16:22:32.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2503, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 16:22:34.776 +0200
17	2025-10-04 16:21:32.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2101, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 16:21:39.884 +0200
18	2025-10-04 16:19:32.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2076, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 16:19:40.053 +0200
19	2025-10-04 16:16:23.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2385, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 16:16:23.771 +0200
20	2025-10-04 16:12:23.000	xiao-esp32-01	CRT_HIGH	warning	{"crt_ms": 2388, "device_id": "xiao-esp32-01", "patient_id": "anon-001"}	2025-10-04 16:12:25.993 +0200

Figura 50: Registro de alertas generadas en tiempo real en la tabla trc_alert. Fuente: elaboración propia.

Para ello, se realizan las siguientes evoluciones:

1. Actualizar el firmware de Arduino con la lógica de generación de datos clínicos de prueba (Anexo U).

2. Desarrollar un nuevo job de Spark Structured Streaming (tfm_stream_enriched.py), para calcular agregados y evaluar reglas (Anexo V).
3. Crear las nuevas tablas en PostgreSQL (trc_observation_minute, trc_observation_5min, trc_alert) y sus índices (Anexo W).
4. Definir consultas SQL para comprobar la ingestión, agregación y generación de alertas (Anexo X).

Esta ampliación supone un paso más allá respecto a la ingesta básica, ya que el sistema deja de limitarse a simplemente registrar los eventos brutos y pasa a transformar el flujo de datos en información procesada, interpretable y explicable. Las reglas de alerta aportan trazabilidad y explicabilidad a las decisiones del modelo de clasificación desarrollado posteriormente, lo que facilita que cada desviación o incidencia pueda asociarse a su causa técnica concreta de forma sencilla.

6.3 Algoritmos de ML e integración embebida

En esta tercera fase del desarrollo se desarrolla e incorpora al sistema un módulo de ML que tiene el objetivo de mejorar la fiabilidad del sistema de monitorización y reducir la generación de alarmas innecesarias.

El sistema desarrollado hasta ahora y que está formado por el dispositivo de medición, la transmisión MQTT y el procesamiento en Spark permite monitorizar en tiempo real el TRC y generar alertas automáticas basadas en umbrales. No obstante, si el sistema dependiera únicamente de estas reglas para generar alarmas clínicas, se correría el riesgo de producir un número elevado de falsos positivos, tal y como se desarrolla en el Capítulo 2. Esto supondría una sobrecarga de alarmas en la UCI, una disminución de la atención del personal sanitario y una mayor tasa de errores, siendo esta una problemática ya ampliamente reconocida en este tipo de contextos clínicos.

Por este motivo, se desarrolla un modelo de ML que permite determinar de manera inteligente las mediciones técnicamente válidas y filtrar las alertas clínicas para que solo se activen cuando la calidad de la señal es adecuada. El modelo se entrena con datos procedentes de mediciones reales realizadas con el dispositivo en distintas condiciones técnicas. Su propósito no es obtener resultados clínicos, sino simular escenarios de uso representativos y reproducir variaciones de iluminación, fuerza o estabilidad que afecten a la fiabilidad de la mediciones.

De esta manera, la utilidad del modelo es funcionar como una capa de validación inteligente que filtra las mediciones y reduce la probabilidad de falsas alarmas sin reemplazar las reglas ya definidas. Estas reglas calculadas en Spark mantienen su utilidad explicativo, ya que cuando el modelo identifica una señal de baja calidad, las alertas basadas en umbrales indican qué factor técnico está provocando el problema, lo que permite al profesional sanitario corregir lo que está fallando en el procedimiento.

Dado que esta decisión debe tomarse durante la propia medición, el modelo se implementa embebido directamente en el microcontrolador ESP32-C3. Esta implementación local del modelo, basada en los principios de *TinyML*, permite ejecutar la inferencia directamente en el microcontrolador con un consumo mínimo de recursos, una latencia prácticamente nula y proporciona un feedback inmediato al usuario mediante un elemento visual en el propio dispositivo. Así, el clínico puede saber en tiempo real si la medición ha sido válida o si debe repetirse sin tener que hacer uso de dispositivos externos como la visualización en un monitor o a la revisión posterior de los datos en el dashboard.

6.3.1 Obtención y preparación de los datos de entrenamiento y test

Para el desarrollo y validación del modelo de *ML* se recopilan datos experimentales obtenidos mediante el propio dispositivo de medición. Las mediciones se realizan en seis voluntarios no clínicos y en un entorno controlado, con el objetivo de reproducir distintas condiciones técnicas de uso. Cada voluntario realiza aproximadamente mediciones para contar finalmente con un total de 150 registros experimentales. Lo datos regidos se adjuntan junta al proyecto en el archivo *"Mediciones sujetos CRT".*

Todas las pruebas se llevan a cabo con consentimiento informado y sin registrar ningún dato personal ni clínico. El objetivo de esta fase no es obtener valores clínicos reales, sino reproducir condiciones técnicas representativas del dispositivo para poder crear una base de datos que permita evaluar el funcionamiento del modelo.

En la práctica, se presentan tres escenarios en relación con la iluminación:

• **Iluminación adecuada:** el dedo se coloca correctamente alineado y el LED integrado en el sensor de color proyecta la luz de forma uniforme.

• **Iluminación excesiva:** el dedo se acerca más de lo recomendado produciendo una sobreexposición por la reflexión del LED.

• **Iluminación insuficiente:** el LED queda parcialmente cubierto o desalineado, por lo que se reduce la cantidad de luz que llega al detector.

En cuanto a la fuerza de compresión, se realizan mediciones variando intencionadamente la presión ejercida durante la medición:

• **Fuerza óptima:** entre 1 y 2 N mantenida durante los 4 segundos establecidos en el protocolo de medición.

• **Fuerza insuficiente:** por debajo de 0.7 N.

• **Fuerza excesiva:** superior a 2.8 N.

Además, se introducen de forma controlada pequeños movimientos o variaciones en la estabilidad del dedo durante la compresión para poder registrar señales de distinta calidad y simular ruido o artefactos.

Durante cada medición, el sistema Arduino recoge las señales de ambos sensores y almacena las siguientes variables que posteriormente se emplean en el proceso de extracción de características y en el entrenamiento del modelo:

• **force_peak (N):** valor máximo de la fuerza aplicada durante la compresión.

• **force_mean_hold (N):** fuerza media mantenida durante los 4 s de compresión.

•**force_hold_ms (ms):** duración de la compresión.

• **lux_avg (lx):** luminosidad media medida por el sensor de color durante la compresión y recuperación.

• **g_baseline / g_min:** valores del canal verde, el cual es el empleado a la hora de medir el TRC, antes de la compresión y en su mínimo durante la compresión. Son empleados para normalizar la curva.

• **curve_monotonicity (0–1):** grado de recuperación de la señal. Indica estabilidad y ausencia de oscilaciones.

• **early_recovery_slope:** pendiente media de la recuperación en los primeros 0.5–0.8 s tras el fin de la compresión. Representa la velocidad inicial del relleno capilar.

• **residual_at_2s:** valor residual de la curva a los 2 s tras la liberación. Indica la parte de la coloración que todavía no se ha recuperado.

• **sqi (0–1):** índice de calidad de la señal, calculado en función de su estabilidad y consistencia a lo largo de la medición.

• **crt_ms (ms):** TRC determinado automáticamente por el algoritmo y definido como el tiempo hasta alcanzar el 90 % de la coloración inicial (t_{90}). (31)

Para clasificar las mediciones se utiliza una medición de referencia realizada en condiciones estandarizadas. A partir de esta, se repiten las mediciones introduciendo variaciones controladas en factores como la fuerza, la iluminación o la estabilidad. La variable de etiqueta asignada a cada medición se determina según la diferencia relativa del TRC (crt_ms) y el índice de calidad de la señal (sqi). Las mediciones con variaciones inferiores al 10 % respecto a la referencia y con sqi > 0.85 se consideran “OK”, las que presentan desviaciones moderadas (10–25 %) o un sqi entre 0.70 y 0.85 se clasifican como “BORDERLINE” y las que tienen desviaciones superiores al 25 % o sqi < 0.70 se etiquetan como “REPEAT”.

Esto permite reproducir de forma controlada las diferentes condiciones técnicas de uso del dispositivo y tener una base realista para el entrenamiento y la validación del modelo de clasificación.

6.3.2 Análisis exploratorio del conjunto de datos

Antes de generar el conjunto sintético y entrenar el modelo de clasificación, es necesario realizar un análisis exploratorio de los datos experimentales para confirmar la coherencia de las

mediciones y la relevancia de las variables registradas. El código utilizado, junto con las figuras generadas en Python que muestran los resultados de este análisis, se incluye en el archivo adjuntado "*Análisis_exploratorio_de_los_datos_muestreados*". En dicho documento se recogen además otras pruebas complementarias realizadas durante la fase exploratoria.

El conjunto de datos cuenta con 150 registros y las variables numéricas se encuentran dentro de los rangos definidos por el sistema. En la Figura 51 se observa la distribución de las clases de calidad (OK, BORDERLINE y REPEAT), que presenta un desbalance natural, el cual se mantiene de forma intencionada para reflejar la variabilidad real del proceso y se tendrá en cuenta en las fases posteriores de ampliación del conjunto de datos y entrenamiento del modelo.

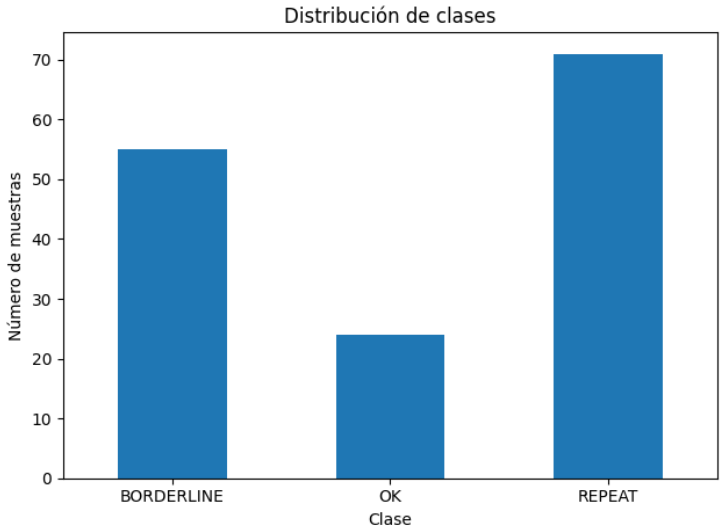


Figura 51: Distribución de clases en el conjunto de datos muestreados. Fuente: elaboración propia.

El análisis de correlaciones muestra relaciones coherentes entre las variables. Se observa que una mejor calidad de señal y una mayor monotonicidad de la curva se relacionan con pendientes de recuperación más pronunciadas y con un menor valor residual a los dos segundos. Esto confirma la coherencia física del sistema y la validez de las mediciones.

En la Figura 52 se representan las distribuciones de las variables sqi y early_recovery_slope para cada clase de calidad. Se puede observar que las mediciones etiquetadas como OK presentan valores más altos tanto de sqi como de early_recovery_slope, lo cual refleja una recuperación más rápida tras el final de la compresión. Las clases BORDERLINE y REPEAT muestran valores progresivamente menores en ambas variables, lo que evidencia una disminución en la calidad de la medición. De esta manera se vuelve a reforzar la coherencia del etiquetado y confirma que las variables seleccionadas permiten distinguir de manera consistente entre mediciones válidas e inválidas.

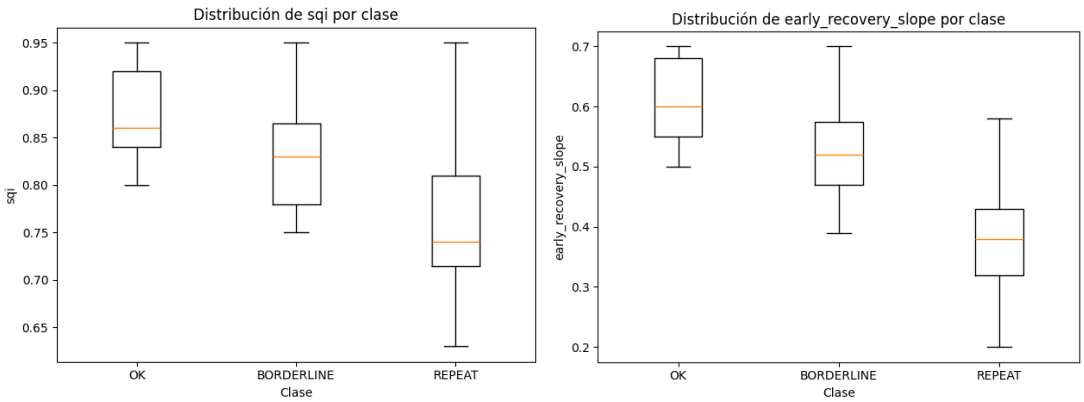


Figura 52: Distribución de las variables sqi y early_recovery_slope por clase de calidad. Fuente: elaboración propia.

6.3.3 Generación del conjunto sintético y etiquetado de las mediciones

A partir de los datos recogidos mediante las mediciones realizadas con el dispositivo se procede a generar una ampliación del conjunto de datos de forma sintética con el objetivo de disponer de un número suficiente de registros para entrenar y validar el modelo de clasificación. El código correspondiente con este proceso y el análisis exploratorio posterior se encuentra en el archivo adjuntado "*Generación_dataset_500_y_análisis_exploratorio*".

La ampliación se lleva a cabo reproduciendo las distribuciones originales y manteniendo las relaciones estadísticas entre variables. Para ello, se emplea un modelo basado en cópulas gaussianas, una técnica que permite describir dependencias multivariantes sin asumir una forma concreta para cada variable. El uso de cópulas gaussianas para la generación de datos sintéticos es una técnica estadística ampliamente empleada en contextos como el del presente proyecto donde se busca preservar dependencias multivariantes entre variables observadas. (50, 51)

En la práctica, se crea una cópula independiente para cada clase de la variable etiqueta de calidad (*OK*, *BORDERLINE* y *REPEAT*) a partir de los datos reales. El procedimiento consiste en transformar las variables originales a un espacio normalizado, estimar la matriz de correlaciones entre ellas y finalmente generar nuevos registros siguiendo esa estructura. Los valores generados se transforman de nuevo al espacio original haciendo uso de la inversa de la distribución empírica de cada variable, por lo que se garantiza que las magnitudes sintéticas tienen los rangos físicos y las formas de distribución de las mediciones reales.

Además, también se aplican restricciones físicas y reglas de coherencia para poder asegurar que los valores generados sean coherentes desde el punto de vista técnico. Por ejemplo, se impone que el valor mínimo del canal verde sea siempre inferior al valor basal (g_min < g_baseline – 50), que la fuerza aplicada esté dentro de los límites del dispositivo (0.6–3.2 N) y que la duración de la compresión no supere el rango permitido (3.3–4.7 s).

Cada registro sintético se etiqueta automáticamente aplicando las mismas reglas que en los datos experimentales y las muestras en las que la etiqueta generada no coincide con la clase objetivo son descartadas.

Finalmente, se añaden metadatos de identificación (subject_id, device_id, marcas de tiempo y notas descriptivas) y se combinan los registros reales con los sintéticos en un conjunto equilibrado de 500 observaciones.

6.3.4 Análisis exploratorio del conjunto de datos sintético

Antes de realizar el entrenamiento del modelo de clasificación, se realiza un análisis exploratorio para verificar la coherencia técnica y estadística del conjunto de datos formado por los 150 registros experimentales y los 350 registros sintéticos.

En la Figura 53 se muestra la distribución de clases del conjunto final, la cual muestra un equilibrio exacto entre las categorías OK, BORDERLINE y REPEAT. Esto garantiza que el modelo no introduzca sesgos hacia una clase determinada durante el proceso de entrenamiento.

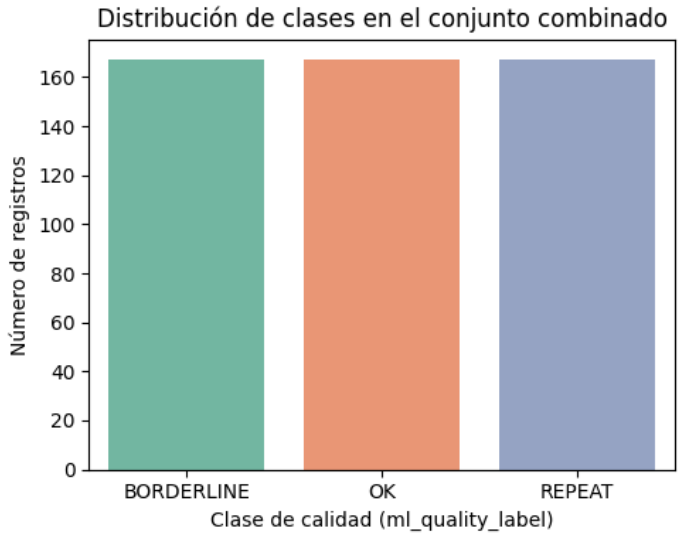


Figura 53: Distribución de clases de calidad en el conjunto combinado (OK, BORDERLINE y REPEAT). Fuente: elaboración propia.

A continuación, se analizan las principales variables técnicas mediante diagramas de caja (Figura 54). Las mediciones etiquetadas como OK presentan valores de fuerza aplicada (force_mean_hold) próximos al rango óptimo de 1–2 N, una duración de compresión estable alrededor de 4 s (force_hold_ms) y un índice de calidad de señal (sqi) superior al de las demás clases. Por otro lado, las clases BORDERLINE y REPEAT tienen una mayor dispersión y desviaciones en fuerza, iluminación o estabilidad, lo que confirma la correcta separación entre categorías de calidad.

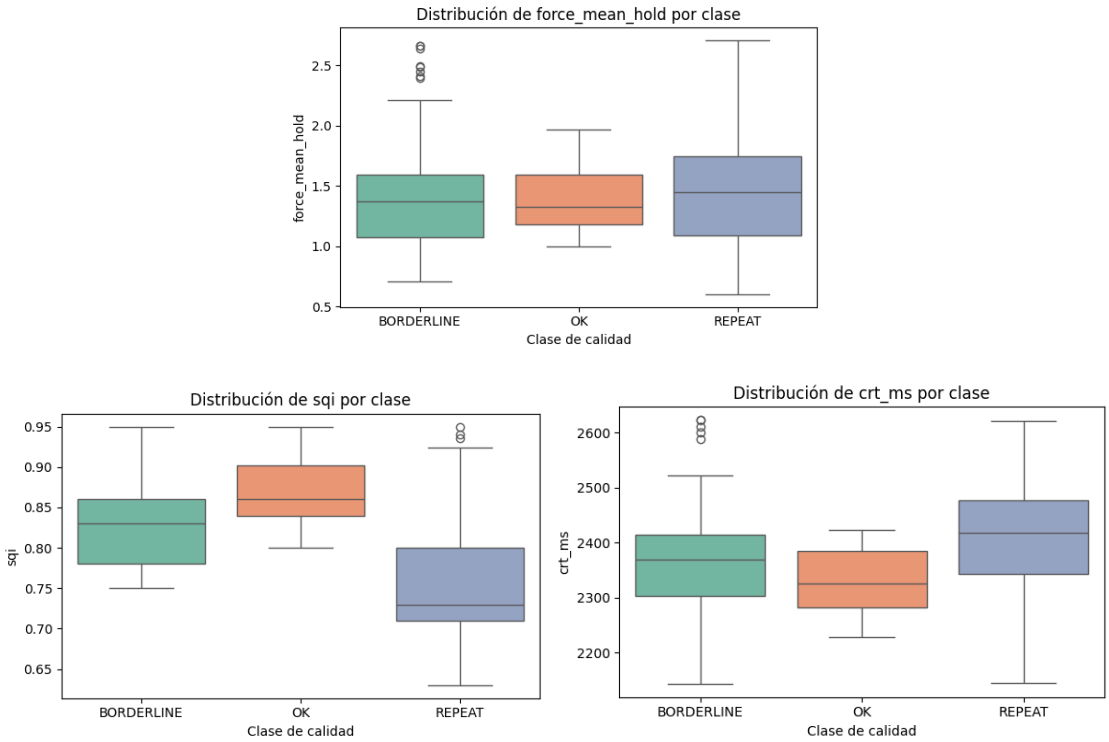


Figura 54: Distribución de las variables técnicas y fisiológicas por clase: fuerza media aplicada (force_mean_hold), índice de calidad de señal (SQI) y TRC (CRT). Fuente: elaboración propia.

La matriz de correlaciones (Figura 55) muestra una relación inversa moderada entre la fuerza media aplicada y el TRC (crt_ms), con un coeficiente de correlación de aproximadamente –0.5. Este patrón es coherente e igual al que se presenta en los datos experimentales, ya que una mayor fuerza de compresión suele asociarse a un vaciado capilar más completo y, por lo tanto, a un tiempo de recuperación más corto. Asimismo, se observan correlaciones positivas entre las métricas de calidad (sqi, curve_monotonicity) y la consistencia temporal de la medición.

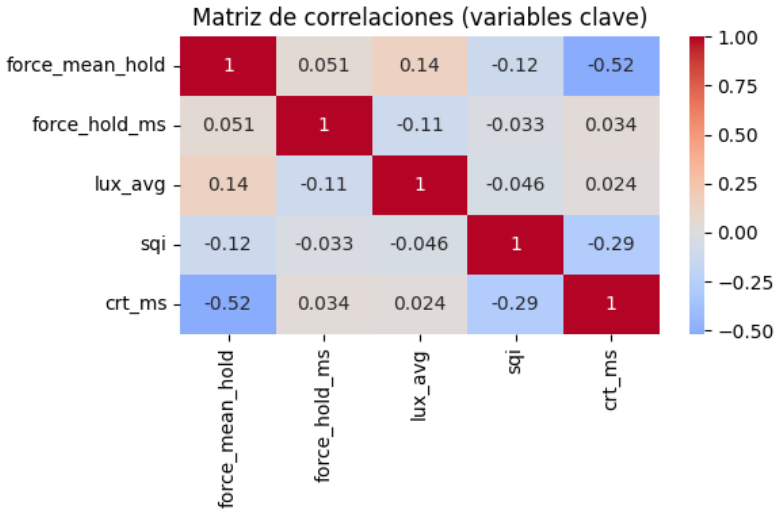


Figura 55: Matriz de correlaciones entre las principales variables técnicas del conjunto combinado. Fuente: elaboración propia.

Además, en la Figura 56 se compara la distribución de clases en los conjuntos real y sintético. En el conjunto original (*raw*), las categorías están desbalanceadas, existiendo una mayor proporción de mediciones *REPEAT* y *BORDERLINE* frente a *OK*. En cambio, el conjunto sintético (*raw_synth*) introduce nuevos registros de forma intencionada para equilibrar las tres clases. Esto evita tener sesgos durante el entrenamiento del modelo de clasificación.

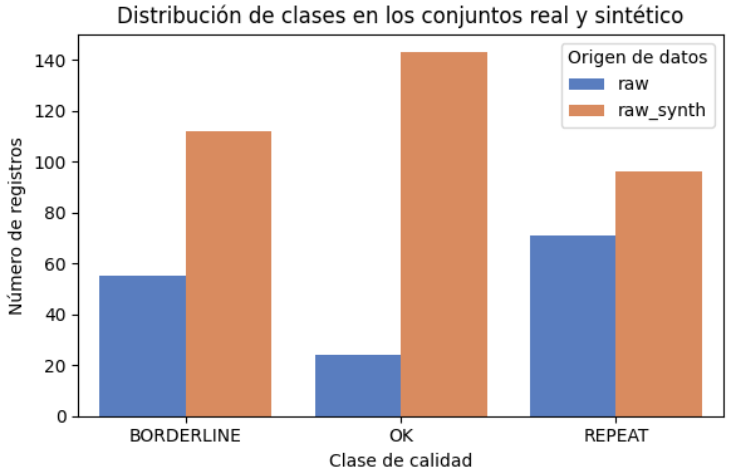


Figura 56: Comparativa del número de registros por clase entre los conjuntos real y sintético. Fuente: elaboración propia.

Por otro lado, las curvas de densidad de las variables force_mean_hold, sqi y crt_ms (Figura 57) muestran un solapamiento casi completo entre los conjuntos real y sintético. Esto demuestra una gran similitud entre los conjuntos y confirma que el proceso de generación sintética mantiene las características estadísticas de las mediciones originales sin introducir sesgos ni distorsionar la variabilidad real de las mediciones.

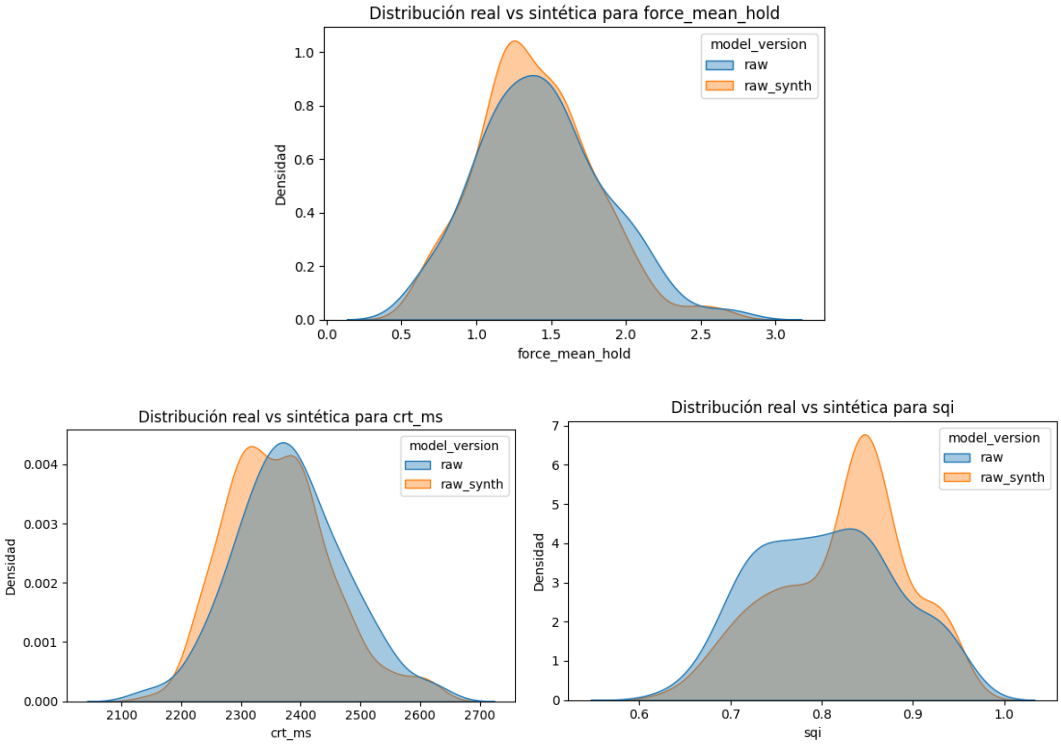


Figura 57: Comparación de las distribuciones reales y sintéticas para las variables fuerza media aplicada (force_mean_hold), índice de calidad de señal (SQI) y TRC (CRT). Fuente: elaboración propia.

6.4.5 Entrenamiento del modelo de clasificación

El objetivo del modelo es clasificar cada medición en una de las tres categorías de calidad establecidas (*OK*, *BORDERLINE* y *REPEAT*) a partir de las variables técnicas y fisiológicas registradas durante la medición. El código correspondiente a esta fase de desarrollo se incluye en el archivo adjuntado *"Entrenamiento_del_modelo"*, donde se detallan los pasos seguidos para el entrenamiento, validación y evaluación del clasificador.

Se decide desarrollar un árbol de decisión como algoritmo base, ya que este permite desarrollar un modelo interpretable, fácilmente integrable en el entorno IoT y con un comportamiento adecuado en datos tabulares como los del actual caso de uso. Este tipo de modelo ofrece además la ventaja de que sus reglas internas pueden representarse de forma explícita, lo que facilita su validación desde el punto de vista técnico y clínico. (52–54) La Figura 58 muestra la estructura final del árbol de decisión entrenado, donde se representan las divisiones y umbrales definidos por las variables técnicas más relevantes.

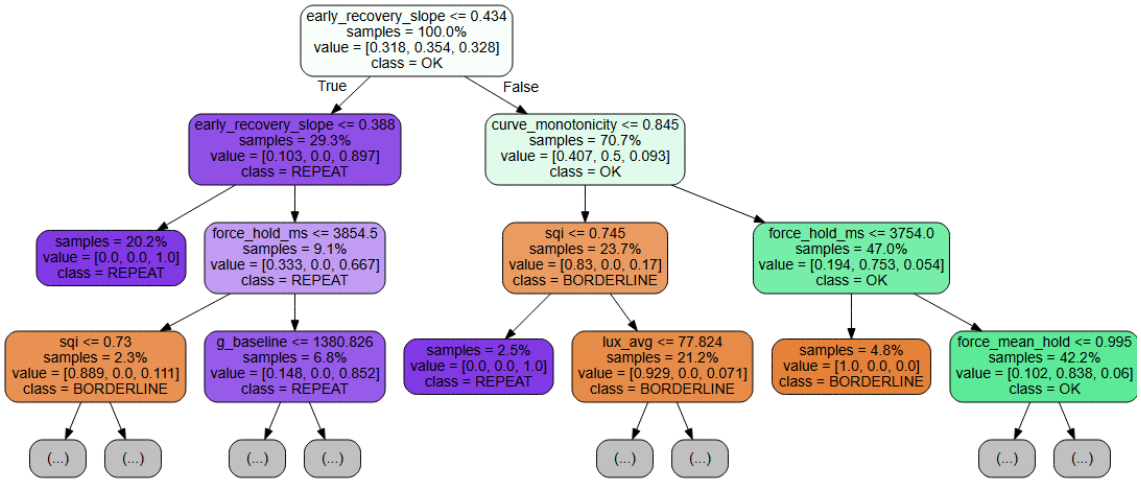


Figura 58: Árbol de decisión resultante tras el entrenamiento del modelo. Fuente: elaboración propia.

Para evitar la pérdida de información entre mediciones de un mismo sujeto, se aplica una validación cruzada agrupada por sujeto (GroupKFold, k = 5). De este modo, todos los registros de a un mismo sujeto se incluyen siempre en el mismo pliegue. Así se garantiza que la evaluación refleja la capacidad del modelo para generalizar entre distintos sujetos. La métrica empleada es el F1-macro, que pondera por igual las tres clases y ofrece una visión equilibrada del rendimiento global.

El proceso de ajuste de los hiperparámetros del modelo se realiza mediante una búsqueda en cuadrícula (*GridSearchCV*), la cual permite explorar combinaciones de profundidad máxima del árbol y tamaños mínimos de división y hoja. Los mejores parámetros obtenidos son: max_depth = 8, min_samples_split = 5 y min_samples_leaf = 1, con un F1-macro medio de **0.948 ± 0.014** entre los cinco pliegues.

La baja desviación estándar obtenida de 1.4 % muestra que existe una alta estabilidad entre sujetos, lo que indica que el modelo mantiene un rendimiento consistente independientemente del grupo de validación. Esto nos sugiere que el árbol de decisión ha aprendido patrones técnicos comunes sin depender de características individuales de cada sujeto.

Aunque el rendimiento medio obtenido es alto, no se observan indicios de sobreajuste, ya que los resultados son homogéneos entre pliegues y las reglas generadas por el modelo son coherentes con las observaciones experimentales. En concreto, el árbol identifica umbrales que resultan fisiológicamente coherentes, como *sqi* < 0.75 o *force_hold_ms* > 4100 ms, los que permiten distinguir correctamente las mediciones válidas de aquellas con una ejecución o una señal deficiente.

Después de realizar el entrenamiento del modelo, se guarda el modelo final junto con los artefactos asociados (codificador de etiquetas, lista de variables, hiperparámetros y reglas del árbol) para su posterior evaluación sobre un conjunto independiente de sujetos no vistos durante el entrenamiento.

6.4.6 Evaluación del modelo y análisis de resultados

La evaluación del modelo se realiza mediante validación cruzada agrupada por sujeto, lo que garantiza que las mediciones de un mismo individuo no se utilicen simultáneamente para el entrenamiento y la validación. El código correspondiente a esta fase de evaluación, junto con las métricas obtenidas, se incluye en el archivo "*Evaluación_del_modelo*".

En este conjunto, el modelo obtiene una exactitud global del 87,6 % y un F1-macro de 0,88, lo que muestra un buen rendimiento y equilibrado entre las tres clases (*OK*, *BORDERLINE* y *REPEAT*). La Tabla de métricas muestra resultados muy similares entre categorías, con F1 individuales de 0,85, 0,97 y 0,85 respectivamente, lo que indica que el modelo mantiene una sensibilidad comparable para detectar tanto mediciones válidas como no válidas.

La Figura 59 se trata de la matriz de confusión obtenida en el test por sujetos. Se puede observar que el modelo identifica correctamente todas de las mediciones *OK* sin producir falsos positivos ni falsos negativos clínicamente relevantes. Las confusiones se concentran entre las clases *BORDERLINE* y *REPEAT*, lo cual resulta lógico, ya que ambas representan mediciones de calidad intermedia o deficiente con límites menos definidos. Se puede afirmar que el modelo discrimina de forma robusta las mediciones técnicamente correctas.

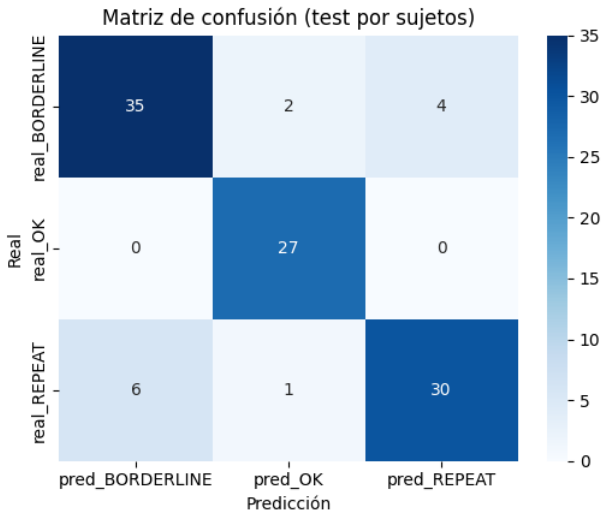


Figura 59: Matriz de confusión del modelo en el conjunto de test. Fuente: elaboración propia.

En cuanto a la interpretabilidad del modelo, la Figura 60 representa la importancia relativa de las variables según el árbol de decisión entrenado. Las características con mayor peso en la predicción son early_recovery_slope, curve_monotonicity y force_hold_ms, seguidas por force_mean_hold y sqi. Este patrón es coherente con los fundamentos técnicos del sistema, ya que las dos primeras variables describen la morfología y estabilidad de la curva de relleno capilar, mientras que las relacionadas con la fuerza y la duración reflejan la correcta realización de la prueba de manera estandarizada.

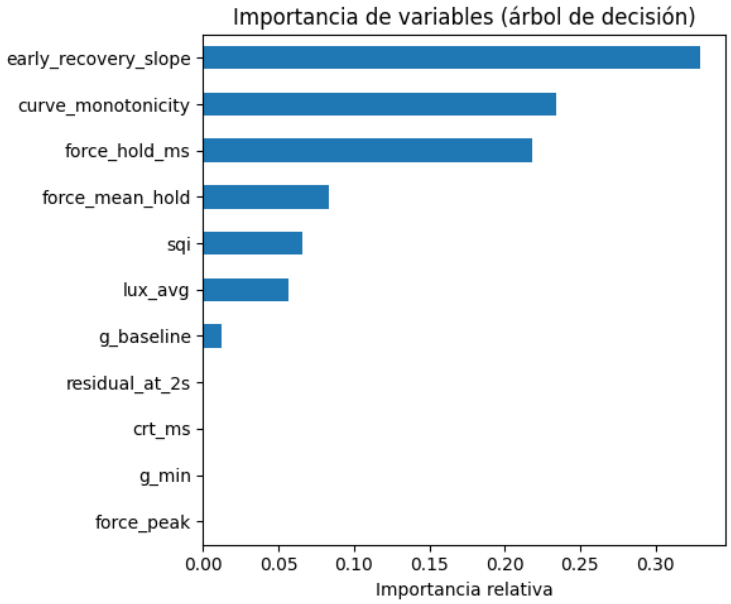


Figura 60: Importancia relativa de las variables en el modelo. Fuente: elaboración propia.

En conclusión, se puede afirmar que las métricas obtenidas muestran un buen equilibrio entre precisión y generalización, y no muestran ningún indicio de sobreajuste. Además, las reglas generadas a partir del modelo son reproducibles y coherentes con las observaciones experimentales, por lo que se puede confirmar su uso como herramienta automatizada para la detección de mediciones de baja calidad en el contexto del sistema IoT desarrollado.

6.3.7 Implementación embebida e integración en el sistema IoT

Tras el entrenamiento y la validación del modelo de clasificación, el siguiente paso es integrarlo en el hardware del sistema para que pueda ejecutar las predicciones directamente en el microcontrolador, sin depender de Python ni de una conexión externa, siguiendo los principios de TinyML mencionados en apartados anteriores. El código correspondiente con esta fase de exportación e integración se incluye en el archivo *"Exportar_ml_tree"*.

Para ello, se realiza una exportación del árbol de decisión desde *scikit-learn* al lenguaje C/C++. Se genera un archivo de cabecera (ml_tree.h) que contiene las estructuras del árbol en forma de arrays (índices de nodos, umbrales, hijos izquierdo y derecho, valores de clase, etc.) y que garantiza una correspondencia exacta con el modelo generado en Python. Este proceso se automatiza mediante un script que extrae la información interna del clasificador y la transforma a un formato compatible con el firmware de Arduino.

El procedimiento de exportación del modelo se basa en la estructura interna del árbol de decisión de *scikit-learn*, siguiendo un enfoque similar al empleado en herramientas como *micromlgen*, que permiten generar código C/C++ a partir de modelos entrenados para su ejecución en microcontroladores. (55)

En la placa, se implementa una función de inferencia ligera (ml_predict) incluida en el archivo ml_inference.h. Esta función recorre el árbol nodo a nodo evaluando las condiciones *if/else* hasta alcanzar una hoja, donde calcula la clase predicha y la probabilidad asociada a partir de los valores almacenados. De este modo, el modelo se ejecuta con una latencia del orden de milisegundos y sin necesidad de librerías externas, lo que lo hace perfectamente adecuado para un dispositivo embebido con recursos limitados.

La inferencia se realiza a partir de un vector de once variables que describen tanto la señal óptica como los parámetros técnicos de la maniobra (force_mean_hold, force_peak, force_hold_ms, lux_avg, g_baseline, g_min, curve_monotonicity, early_recovery_slope, residual_at_2s, sqi y crt_ms). Este orden se mantiene idéntico al utilizado durante el entrenamiento para asegurar la compatibilidad entre ambos entornos.

En cada medición, la placa calcula las variables necesarias, ejecuta el modelo localmente y obtiene una etiqueta de calidad (ml_quality_label, con valores *OK*, *BORDERLINE* o *REPEAT*) junto con una puntuación de confianza (ml_quality_score). Estos resultados se incorporan directamente al mensaje MQTT publicado en el tópico correspondiente, junto con el resto de los parámetros clínicos y técnicos de la medición. De esta forma, el resultado del modelo se transmite y almacena en la base de datos y se visualiza en Grafana junto con las demás señales, manteniendo la coherencia del flujo de datos y la trazabilidad completa del sistema. Además, el firmware implementa un pequeño mecanismo de retroalimentación visual mediante el LED integrado en la placa, que permite al usuario identificar de manera inmediata el resultado de la clasificación (por ejemplo, parpadeo corto para *BORDERLINE* o más largo para *REPEAT*). Esta funcionalidad, aunque sencilla, refuerza la utilidad práctica del dispositivo en escenarios de uso real.

Para verificar la integración del modelo embebido en el flujo de datos, se realizaron pruebas de transmisión y almacenamiento en la base de datos empleando un conjunto reducido de mediciones simuladas. En esta verificación inicial, los valores de los sensores (fuerza aplicada, luminosidad, SQI y TRC) se generaron de manera aleatoria dentro de rangos físicamente plausibles y fueron procesados por el algoritmo embebido en la placa. Las Figuras 61 y 62 muestran el resultado de esta prueba funcional, donde se observa la inserción automática de los valores generados por el modelo en las tablas de observaciones y alertas del sistema, respectivamente. En la tabla trc_observation se registran las etiquetas y puntuaciones de calidad asociadas a cada medición, mientras que en trc_alert se genera un evento *warning* cuando el modelo detecta una medición no válida o repetida.

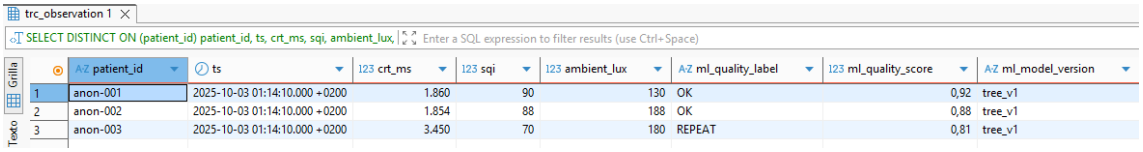


Figura 61: Inserción automática de mediciones procesadas por el modelo en la tabla trc_observation. Fuente: elaboración propia.

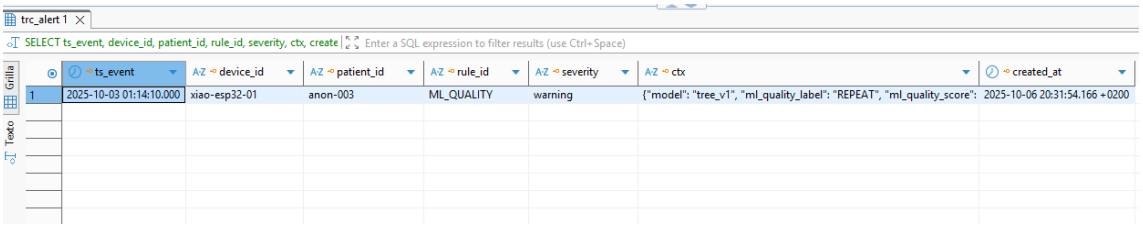


Figura 62: Generación de alerta de calidad del modelo en la tabla trc_alert. Fuente: elaboración propia.

6.4 Visualización clínica y monitorización de datos en tiempo real

Una vez consolidada la infraestructura de adquisición y procesamiento del sistema IoT, el siguiente paso consiste en diseñar un entorno de visualización clínica que permita interpretar la información generada de forma clara, estructurada y útil para la toma de decisiones. En un contexto hospitalario y especialmente en una UCI, el poder disponer de datos en tiempo real solo resulta valioso si estos se presentan de manera clara, ordenada y fácilmente interpretable para el personal sanitario. Es por ello por lo que se desarrolla un conjunto de paneles dinámicos en Grafana, conectados directamente con la base de datos PostgreSQL (ver Figura 63), que transforman los datos obtenidos del dispositivo en información con valor clínico. Además, el dashboard está configurado para actualizarse de forma automática para que las visualizaciones de actualizan en tiempo real al recibir nuevos datos desde la plataforma IoT.

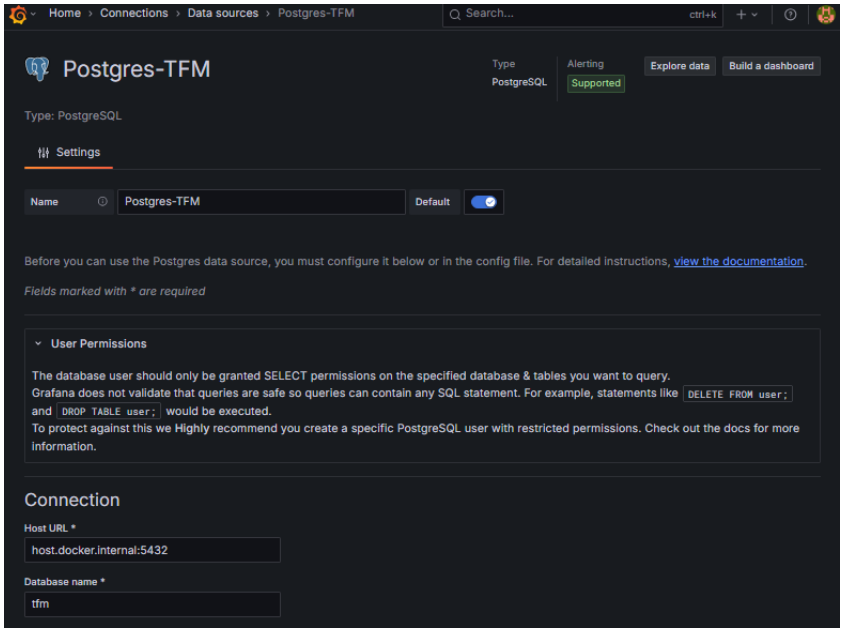


Figura 63: Configuración de la conexión entre Grafana y la base de datos PostgreSQL (Postgres-TFM). Fuente: elaboración propia.

El dashboard diseñado actúa como un módulo de monitorización multipaciente y está estructurado en diferentes niveles de detalle (ver Figura 64). En la parte superior, las visualizaciones muestran una visión global del estado hemodinámico de los pacientes de la unidad, en los niveles intermedios se muestra el estado actual y la evolución temporal del parámetro de CRT y, por último, en la capa inferior se recogen las alertas generadas por el sistema, tanto clínicas como técnicas.

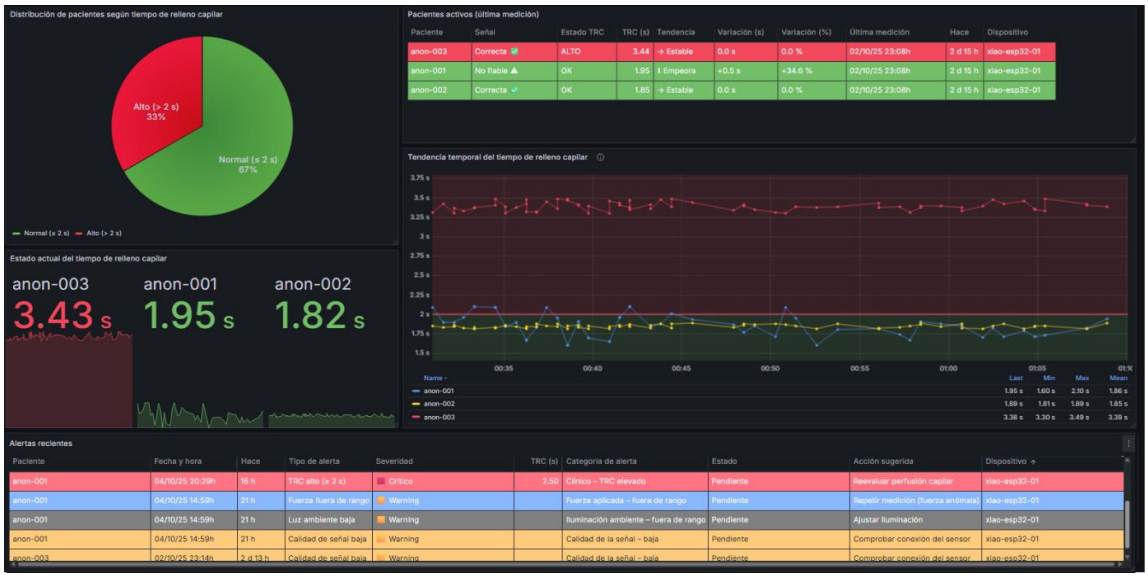


Figura 64: Dashboard completo de monitorización multipaciente desarrollado en Grafana.). Fuente: elaboración propia.

Estas alertas combinan la información procedente del modelo de Machine Learning embebido, que evalúa automáticamente la calidad de cada medición, con las reglas de umbral generadas en Spark que aportan la explicación técnica del motivo de cada aviso. Gracias a esta integración, el sistema no solo muestra los valores registrados, sino que valida la fiabilidad de las mediciones y prioriza la atención en los casos que realmente lo requieren, reduciendo falsos positivos y mejorando la eficiencia del entorno clínico.

De este modo, la interfaz sigue el mismo proceso de análisis que realiza el personal sanitario. Empieza con una de una visión general de la unidad, a continuación pasa al detalle individual de cada paciente y finaliza con la detección de los casos que requieren intervención inmediata.

Para reflejar este flujo de trabajo, el dashboard se organiza en cinco paneles complementarios, diseñados para cubrir las distintas etapas del proceso de monitorización.

- **Panel 1 – Estabilidad global de los pacientes:** representa de forma global el estado de la unidad, mostrando la proporción de pacientes con valores normales (≤ 2 s) y prolongados (> 2 s).
- **Panel 2 – Pacientes activos (última medición):** muestra en formato tabular la información más reciente de cada paciente, incluyendo el valor de CRT, la etiqueta de calidad del modelo y la tendencia.
- **Panel 3 - Estado actual del TRC por paciente:** representa de forma visual y numérica el valor instantáneo de CRT por paciente, utilizando codificación cromática (verde: normal, rojo: prolongado).
- **Panel 4 - Tendencia temporal del TRC:** permite visualizar la evolución del parámetro y detectar variaciones progresivas o deterioros incipientes.
- **Panel 5 - Alertas recientes:** recoge tanto las **alarmas clínicas** como las **alertas técnicas de calidad**, integrando la información del modelo de Machine Learning y las explicaciones generadas por los umbrales de Spark.

Las consultas SQL utilizadas para la construcción de cada panel en Grafana se incluyen en el Anexo Y.

6.4.1 Principios de diseño de la interfaz de monitorización

Además del desarrollo funcional del sistema, el diseño visual del dashboard se ha planteado con un enfoque centrado en el usuario clínico, priorizando la claridad perceptiva y la rapidez de interpretación en el entorno asistencial.

El diseño del dashboard se ha planteado con un enfoque de usabilidad clínica, procurando que la interfaz sea intuitiva y comprensible incluso en entornos de alta carga asistencial. Para ello, se aplican principios de diseño centrado en el usuario sanitario:

- **Jerarquía perceptiva:** como se ha mencionado anteriormente, la disposición de los paneles sigue el flujo habitual de análisis clínico. Este orden guía al usuario desde la

observación general hasta la acción clínica, tal como ocurre en el proceso de análisis habitual del personal sanitario.

- **Coherencia cromática:** se emplean colores reconocibles en el entorno hospitalario, como el verde para valores normales, rojo para situaciones críticas y amarillo para condiciones de precaución, lo que facilita una interpretación inmediata y reduce el riesgo de error visual.
- **Equilibrio visual:** se combinan gráficos y tablas de forma armonizada para mantener una buena legibilidad sin sobrecargar la interfaz. Este equilibrio permite acceder rápidamente a la información sin perder contexto ni detalle. Además, se emplea un fondo oscuro típicamente empleado en monitores clínicos, el cual mejora el contraste visual y facilita la lectura.
- **Legibilidad de valores clave:** los parámetros más relevantes, como el TRC y la calidad de la señal, se muestran con mayor tamaño y contraste, siguiendo la jerarquía visual de los monitores clínicos. Esto facilita llevar la atención a los datos que necesitan una supervisión prioritaria.

6.4.2 Panel 1: Estabilidad global de los pacientes

Este primer panel mostrado en la Figura 65, permite tener al clínico una visión del estado hemodinámico global de los pacientes monitorizados. El principal objetivo del panel es mostrar de forma sencilla la proporción de pacientes con un TRC dentro o fuera del rango clínicamente normal (≤ 2 s), permitiendo valorar rápidamente la situación general de la unidad, priorizar la atención según el nivel de riesgo o establecer la redistribución de recursos asistenciales en función del número de pacientes inestables.

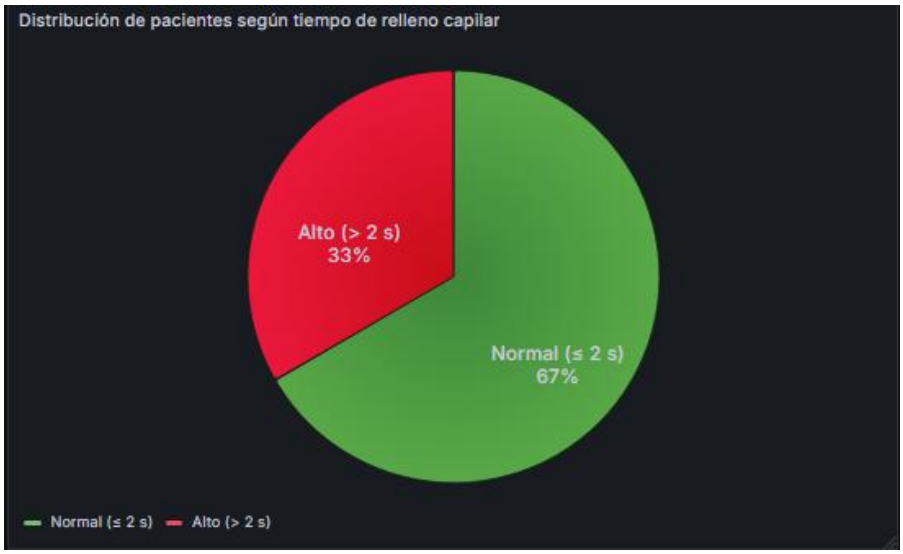


Figura 65: Panel 1: Estabilidad global de los pacientes. Fuente: elaboración propia.

Desde el punto de vista técnico, el panel emplea los datos de la última medición de cada paciente para que el análisis refleje el estado actual y no un promedio histórico del paciente. El sistema selecciona automáticamente la lectura más reciente de TRC y solo considera aquellas mediciones clasificadas como válidas (ml_quality_label = OK) por el modelo de Machine Learning embebido. Este filtro asegura que la representación se base únicamente en mediciones válidas, lo que evita tener datos no fiables.

Esta elección es especialmente importante en el entorno de CI, en el que las condiciones hemodinámicas de los pacientes críticos cambian constantemente y la fiabilidad del dato es tan relevante como su valor absoluto.

En cuanto al diseño visual, se utiliza una representación tipo pie chart. La paleta de colores escogida es coherente con el resto del dashboard y se ajusta a las convenciones clínicas empleadas para facilitar la interpretación visual:

- **Verde (Normal ≤ 2 s)**: pacientes con valores dentro del rango fisiológico con una perfusión periférica adecuada.
- **Rojo (Alto > 2 s)**: pacientes con TRC prolongado, señal de posible alteración hemodinámica que requiere mayor atención o tratamiento.

El uso de colores contrastados y coherentes con su significado clínico facilita una interpretación rápida y sin ambigüedades, ya que el color actúa como un recurso informativo y no solo como un elemento visual. Además, la proporción de cada grupo se acompaña de su porcentaje, lo que evita tener que realizar cálculos adicionales.

En conjunto, este panel cumple una doble función complementaria:

1. **Clínica**: ofrece un recurso visual y actualizado del equilibrio hemodinámico de la unidad a partir de mediciones validadas por el modelo de ML.
2. **Operativa**: sirve como indicador global para guiar la priorización de revisiones dentro del flujo de monitorización continua.

6.4.3 Panel 2: Paciente activos (última medición)

Este segundo panel desarrollado permite tener una visión consolidada del estado de los pacientes monitorizados. Muestra en una sola tabla resumen la última medición registrada del TRC junto con la información complementaria sobre la calidad de la señal, la tendencia y la variación temporal (véase Figura 66).

Figura 66: Panel 2: Paciente activos (última medición). Fuente: elaboración propia.

Su papel es que el personal clínico pueda reconocer de forma **inmediata de cambios relevantes** de los valores fisiológicos y facilitar la priorización de pacientes, ya que permite distinguir de un vistazo cuáles se encuentran estables y cuáles cuáles muestran deterioro.

Desde el punto de vista técnico, cada fila muestra un paciente activo e integra de forma automática los datos procedentes del sistema IoT, el procesamiento en Spark y el resultado del modelo de **ML embebido**.

Solo las mediciones etiquetadas como "OK" por el modelo se consideran válidas para el análisis clínico para poder asegurar que los valores mostrados vienen de señales correctamente adquiridas.
En la columna **"Señal"** el sistema resume esta información de forma binaria:

- **Correcta:** cuando la medición ha sido validada por el modelo.
- **No fiable:** cuando el modelo detecta una anomalía técnica.

Al pasar el cursor sobre una señal marcada como "No fiable", se despliega un tooltip, como se muestra en la Figura 67, que detalla el motivo específico del problema, obtenido a partir de las alertas automáticas generadas en Spark. Estas alertas que han sido calculadas con umbrales para cada parámetro técnico permiten tener explicabilidad al indicar qué factor está afectando a la medición (por ejemplo, iluminación fuera de rango, fuerza excesiva o calidad de señal baja), lo que ayuda al personal clínico a corregirlo.

Figura 67: Panel 2 mostrando el Tooltip de causa técnica asociado a la calidad de señal. Fuente: elaboración propia.

De esta manera, el modelo determina si la medición es válida y las reglas de Spark explican el porqué en el caso de que no lo sea, por lo que se ofrece una inteligencia automática y también transparencia operativa.

Por otro lado, el estado clínico establecido a partir del TRC se calcula de manera automática a partir del valor más reciente. Los valores ≤ 2,0 s se consideran normales (*OK*), mientras que los superiores a 2,0 s se clasifican como *ALTO* y hacen que la fila completa se muestre de color rojo. Además, el sistema evalúa la evolución reciente del paciente comparando sus dos últimas mediciones. Si la variación es inferior a ±0,05 s o ±0,5 %, se considera "*Estable*", si el valor disminuye se muestra "*Mejora*" y si aumenta "*Empeora*". Estos umbrales se definen para evitar que pequeños cambios fisiológicamente no relevantes se interpreten como cambios importantes. Las flechas direccionales en la columna de tendencia (⬆, ⬇, →) refuerzan la interpretación visual, permitiendo al usuario comprender la evolución del paciente sin detenerse en los valores numéricos.

En cuanto al diseño visual, la tabla emplea una estructura compacta y ordenada que permite visualizar de forma simultánea a varios pacientes. Los casos con TRC alto o con mediciones más recientes se sitúan automáticamente en primer lugar para priorizar a los pacientes potencialmente inestables.

Al igual que en la visualización anterior, la codificación cromática mantiene la coherencia clínica utilizando verde para normalidad, rojo para valores elevados y amarillo para condiciones de precaución.

6.4.4 Panel 3: Estado actual del TRC por paciente

El tercer panel (Figura 68) desarrollado transforma los valores de TRC en una lectura visual inmediata y dsitingue mediante colores los pacientes dentro del rango normal y aquellos con valores prolongados. Mientras que el panel anterior muestra la última medición y la evolución inmediata de cada paciente, este se centra en la lectura puntual del valor actual. Está diseñado para priorizar la rapidez de interpretación y la claridad visual propias de los sistemas de monitorización utilizados en entornos de UCI.

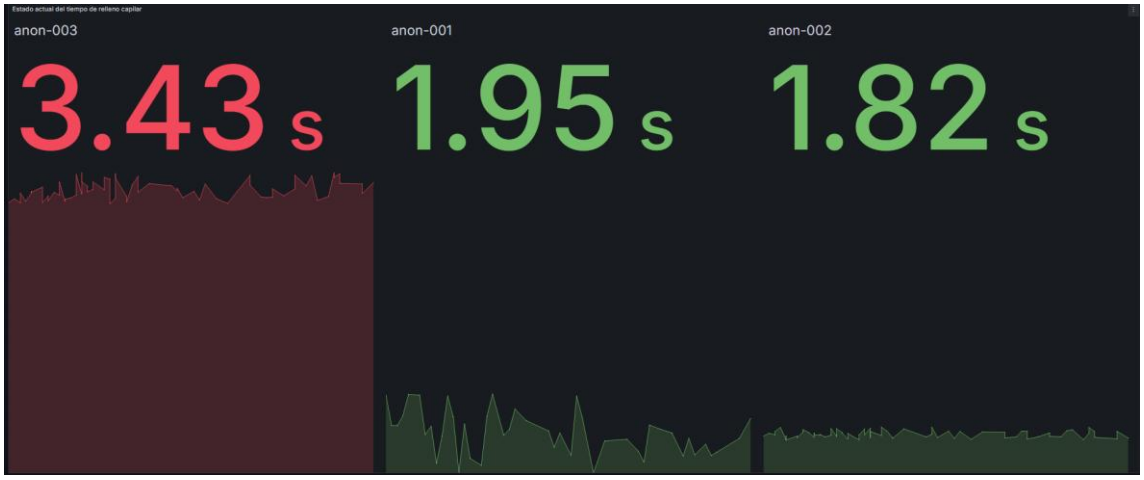


Figura 68: Panel 3: Estado actual del TRC por paciente. Fuente: elaboración propia.

El panel se construye mediante un elemento de tipo Stat que consulta el último valor de TRC válido registrado por cada paciente dentro del rango temporal seleccionado. Para garantizar la fiabilidad de los datos, solo se representan mediciones clasificadas como OK por el modelo de ML.

En cuanto al diseño, se aplica el patrón de *Big Numbers*, habitual en entornos críticos, que resalta los valores principales en un formato grande y de alto contraste para facilitar su lectura inmediata.

Cada tarjeta muestra el identificador del paciente, el valor actual del TRC expresado en segundos y un el mismo código de color semántico empleado en los anteriores paneles para identificar de un vistazo el nivel de riesgo. Los pacientes se ordenan de izquierda a derecha según severidad, por lo que los casos críticos aparecen primero.

Debajo de cada valor numérico se incluye una minigráfica de tendencia delTRC que añade contexto visual sin sobrecargar la interfaz y permite distinguir rápidamente si un paciente se mantiene estable o si ha presentado fluctuaciones relevantes en los últimos minutos.

6.4.5 Panel 4: Tendencia temporal del TRC

El cuarto panel (Figura 69) muestra la evolución temporal del TRC para los diferentes pacientes monitorizados. Este elemento del dashboard resulta crucial a nivel clínico, ya que facilita la detección de patrones, variaciones progresivas o posibles deterioros hemodinámicos. De esta manera, complementa la información de los paneles previos y proporciona una información dinámica.

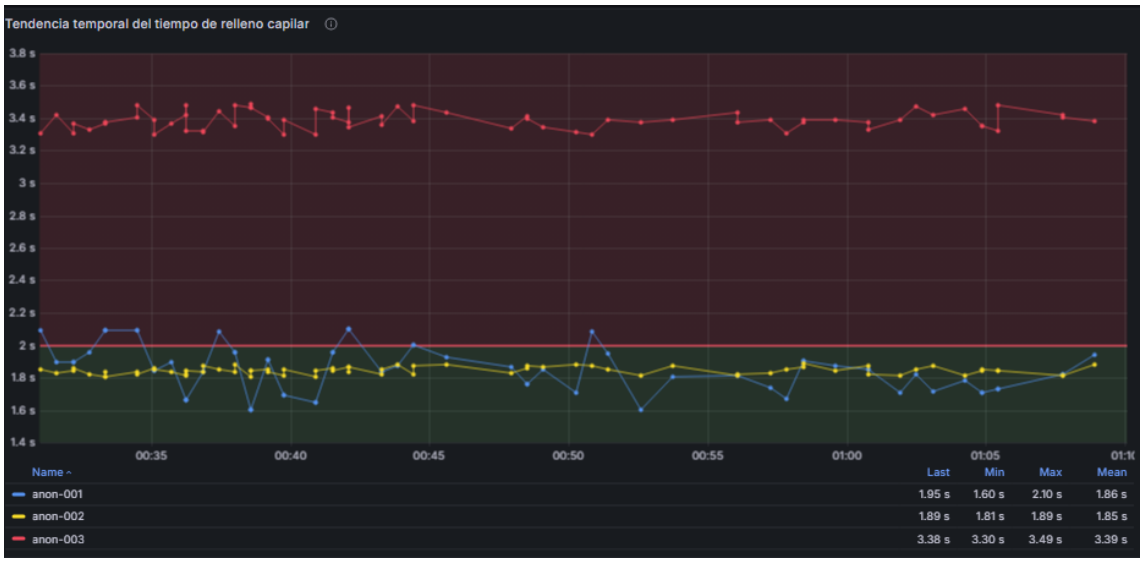


Figura 69: Panel 4: Tendencia temporal del TRC. Fuente: elaboración propia.

Como en las anteriores visualizaciones, solo se incluyen señales validadas por el modelo de Machine Learning embebido (ml_quality_label = OK). Las series se actualizan automáticamente dentro del rango temporal seleccionado para que los datos mostrados reflejen siempre las mediciones más recientes.

El umbral clínico de 2 segundos, que marca el límite entre la normalidad y la posible hipoperfusión, se representa mediante una línea horizontal que separa dos franjas de color: verde por debajo del valor normal y roja por encima. La escala del eje vertical se ajusta de forma automática en función del rango de valores del intervalo, por lo que se asegura una visualización correcta incluso cuando se introducen nuevos pacientes o mediciones con distintos intervalos de TRC.

Además, también se incluye una leyenda en formato de tabla que muestra información cuantitativa resumida del TRC para cada paciente en el intervalo de tiempo seleccionado (valor mínimo, máximo, medio y último valor registrado), complementando la lectura visual y ofreciendo una referencia numérica fácil de interpretar.

6.4.6 Panel 5: Alertas recientes

El último panel desarrollado permite visualizar en tiempo real y de forma centralizada las alertas generadas por el sistema de monitorización (Figura 70). Su objetivo es ofrecer una visión clara, rápida y accionable de las incidencias detectadas por el sistema IoT, incluyendo tanto las de naturaleza clínica asociadas al TRC como las de la calidad de medición.

Figura 70: Panel 5: Alertas recientes. Fuente: elaboración propia.

En la práctica clínica, esta tabla funciona como un panel operativo de apoyo a la decisión que permite al equipo asistencial identificar de un vistazo las alertas activas o más recientes y distinguir su tipo, gravedad y estado. El orden de presentación de las alertas es el siguiente:

1. Las alertas **"Pendientes"** se muestran siempre en primer lugar, priorizando la lectura de las que todavía no han sido atendidas/solucionadas.
2. Dentro de estas, las alertas **clínicas críticas** (tiempos de relleno capilar ≥ 2 s) se sitúan por encima del resto, al ser más críticas y requerir de atención más inmediata.
3. Finalmente, el orden cronológico descendente garantiza que la información esté actualizada y favorece la detección de **episodios agudos o repetitivos**.

Este orden se alinea con la dinámica y flujos de trabajo de las UCI donde la rapidez en la identificación de alteraciones es crítica para la toma de decisiones. Así, el panel no solo muestra los eventos, sino que los organiza según su prioridad clínica real, reduciendo la carga cognitiva del usuario y haciendo que su respuesta e intervención sean más rápidas.

En cuanto a la codificación visual, se han aplicado los siguientes criterios que ayudan a los clínicos a interpretar la gravedad y el contexto de las alarmas de un solo vistazo.:

- Las alertas **clínicas de** TRC elevado se etiquetan como **"Crítico"** y utilizan tonos rojos para resaltar su urgencia.
- Las alertas de **calidad de la señal** (por ejemplo, luz insuficiente, fuerza fuera de rango o señal baja) se marcan como **"Warning"**, usando colores secundarios (amarillos, azules o grises par cada tipo) que mantienen la diferenciación sin generar saturación visual.

Cada registro muestra información relevante del evento:

- **Fecha y hora** del evento y **tiempo transcurrido** ("Hace"), lo que evita tener que hacer cálculos adicionales.
- **Identificación del paciente y del dispositivo** para contextos de monitorización multipaciente.
- **Tipo y categoría de alerta** para diferenciar entre las clínicas y técnicas.
- **Valor de TRC (s)** cuando aplica.
- **Estado** ("Pendiente" o "Reconocida").
- **Acción sugerida**, que proporciona una recomendación inmediata ("Reevaluar perfusión capilar", "Ajustar iluminación", "Repetir medición").

Las columnas con campos temporales y de identificación se sitúan al inicio de la tabla (para priorizar y contextualizar el evento) y están acompañados a continuación de los aspectos de diagnóstico y finalmente los de resolución (estado y acción). Este orden sigue un flujo lógico, desde la detección hasta la intervención, para facilitar la priorización de la información especialmente en contextos de alta carga asistencial.

Capítulo 7. DISCUSIÓN

El desarrollo del sistema ha permitido comprobar la viabilidad técnica de una solución embebida e inteligente para la medición automática del TRC, integrando en una misma arquitectura la adquisición de datos, el procesamiento local, la conectividad IoT y la visualización clínica en tiempo real. Los resultados obtenidos demuestran que el sistema puede funcionar de manera autónoma y estable, y que las tecnologías embebidas y de *machine learning* pueden aplicarse de forma adecuada en entornos asistenciales críticos como las UCI.

Utilidad y adecuación de la metodología

La metodología híbrida basada en Stage-Gate ha resultado adecuada para un proyecto con componentes tan distintos entre sí como el hardware, el software, la conectividad IoT y la analítica de datos. Trabajar por fases permitió avanzar de forma estructurada, validar cada módulo por separado y mantener la coherencia global del sistema. Al mismo tiempo, incorporar elementos de metodologías ágiles fue clave para poder adaptarse a los ajustes necesarios durante el desarrollo, especialmente en la configuración del flujo IoT.

Resultados obtenidos

Como se ha mencionado en anteriores capítulos, este proyecto se enmarca en la línea de investigación desarrollada por el grupo IASalud, que en su etapa inicial se centró en el diseño y validación del dispositivo de medición del TRC. En esta nueva fase se ha avanzado significativamente, consiguiendo no solo cumplir los objetivos planteados, sino también superar las expectativas iniciales al implementar una arquitectura completa que abarca todas las capas del sistema, desde el procesamiento embebido hasta la ingesta, el almacenamiento y la visualización clínica en tiempo real. Este desarrollo supone una contribución directa y valiosa para la línea de investigación, al proporcionar una base tecnológica sólida y extensible sobre la que se podrán construir futuras fases de validación y despliegue clínico.

Se puede afirmar que el sistema final ha mostrado un funcionamiento estable y coherente entre sus distintos componentes. La comunicación mediante MQTT permite mantener un flujo de datos continuo y de baja latencia, mientras que Spark Structured Streaming asegura la ingesta y el almacenamiento estructurado en tiempo real.

El modelo de TinyML embebido en el microcontrolador permite realizar la inferencia local con un consumo muy bajo de recursos y ofrecer *feedback* inmediato al usuario, lo que supone una mejora clara respecto a los métodos manuales o dependientes de procesamiento externo. En esta etapa no se ha realizado una evaluación específica del rendimiento del modelo en cuanto a latencia o consumo de recursos, por lo que estos parámetros se han estimado tomando como referencia la evidencia descrita en la literatura científica sobre modelos embebidos equivalentes.

Por último, la interfaz de Grafana ha mostrado una representación clara y adaptada al flujo clínico, evidenciando que la información generada por el sistema puede tener un uso real en la monitorización y toma de decisiones clínicas.

En términos generales, los resultados obtenidos no solo demuestran la viabilidad técnica del sistema, sino también su potencial para favorecer un modelo de atención sanitaria más sostenible. La automatización de la medición del TRC contribuye a optimizar recursos asistenciales, reducir la carga de trabajo del personal clínico y mejorar la eficiencia de procesos, especialmente en entornos con recursos limitados. Del mismo modo, el empleo de tecnologías abiertas y componentes de bajo coste refuerza la sostenibilidad económica y la posibilidad de replicar el sistema en distintos contextos hospitalarios.

Desde el punto de vista de la innovación, este proyecto constituye un primer acercamiento documentado en la literatura científica que combina de forma integrada procesamiento embebido, IA ligera y visualización clínica adaptada al flujo de trabajo en UCI para la medición del TRC. Hasta la fecha, los estudios existentes se habían centrado principalmente en el desarrollo del dispositivo y en la sensorización del parámetro, pero sin una explotación real del dato ni la aplicación de técnicas de IA a este tipo de mediciones.

Del mismo modo, no se habían propuesto arquitecturas de datos específicas ni sistemas de visualización clínica que permitieran integrar el TRC como un parámetro dinámico, visualizar tendencias en tiempo real o generar alertas adaptadas a la práctica asistencial. En este sentido, el proyecto representa una aportación claramente original e innovadora, tanto desde la perspectiva técnica como en su aplicabilidad al contexto clínico, abriendo una línea de investigación interesante y potente sobre la digitalización inteligente del TRC.

Limitaciones del proyecto

A pesar de los resultados obtenidos, el proyecto presenta algunas limitaciones que deben tenerse en cuenta para las futuras etapas de desarrollo:

- **Integración de Kafka:** al inicio se contempló incluir Apache Kafka como capa intermedia entre Mosquitto y Spark, pero finalmente se descartó por la complejidad añadida y el poco valor que aportaba en esta fase inicial. La arquitectura actual, basada solo en MQTT y Spark, es más simple y adecuada para la validación técnica, aunque limita su escalabilidad inmediata en entornos con grandes volúmenes de datos.
- **Configuración de la infraestructura IoT:** durante la fase de validación se utilizó un broker público (*test.mosquitto.org*) para facilitar la comunicación MQTT. Aunque esta elección es adecuada para las pruebas técnicas realizadas sin datos sensibles, en un entorno clínico real debería sustituirse por un broker privado autenticado y con comunicación cifrada (TLS) para garantizar la seguridad y confidencialidad de los datos.
- **Almacenamiento analítico en Parquet:** el sistema implementa un *Data Lake* en formato Parquet, pero en esta fase solo se ha utilizado como repositorio de persistencia. Su

explotación analítica o integración en procesos de entrenamiento de modelos con datos reales queda pendiente para fases posteriores del proyecto.

- **Tamaño y naturaleza del conjunto de datos:** el modelo de ML se entrenó con un conjunto experimental limitado. Aunque este enfoque permitió simular diferentes escenarios de calidad de medición, la falta de datos clínicos reales reduce su capacidad de generalización y hace necesario validarlo con registros fisiológicos en el futuro.
- **Evaluación de rendimiento del modelo embebido:** el modelo *TinyML* integrado en el microcontrolador no se ha sometido a pruebas específicas de rendimiento, como la medición de latencia, consumo energético o carga computacional. En esta etapa se ha priorizado la validación funcional, apoyándose en la evidencia existente en la literatura científica sobre la eficiencia de los modelos embebidos de características similares. No obstante, la evaluación experimental del rendimiento real del modelo será un paso necesario en futuras fases del proyecto.
- **Validación clínica:** la validación realizada ha sido de tipo técnico y funcional. No se han realizado pruebas clínicas con pacientes ni una evaluación comparativa con procedimientos manuales, por lo que los resultados deben interpretarse dentro de este alcance. Este punto representa una limitación importante, pero también una oportunidad clara para la continuación del trabajo.
- **Limitaciones del hardware embebido:** el microcontrolador empleado (XIAO ESP32-C3) ofrece una buena relación entre consumo y capacidad, aunque sus recursos son limitados. Esto condiciona la complejidad de los modelos que pueden desplegarse localmente y restringe la posibilidad de implementar arquitecturas más avanzadas sin aplicar técnicas adicionales de optimización.

En conjunto, las limitaciones identificadas no restan valor al trabajo realizado, sino que ayudan a delimitar su alcance y marcan el camino para su evolución futura. El sistema actual constituye una base técnica sólida, capaz de integrarse en entornos de monitorización hospitalaria y de escalar hacia arquitecturas de datos más complejas. La propuesta se ajusta a los requisitos de interoperabilidad y eficiencia de las UCI y representa un paso adelante hacia la digitalización inteligente de parámetros clínicos tradicionalmente manuales.

Capítulo 8. CONCLUSIONES

En este capítulo se presentan las conclusiones generales del trabajo a partir del desarrollo realizado y los resultados obtenidos, así como una reflexión personal sobre la experiencia y el aprendizaje adquiridos a lo largo del proyecto.

8.1 Conclusiones del trabajo

El proyecto ha demostrado la viabilidad técnica y funcional de un sistema embebido e inteligente para la medición automatizada del TRC, confirmando que este parámetro puede digitalizarse y monitorizarse en tiempo real mediante una solución ligera, autónoma y conectada. Los resultados obtenidos muestran que el enfoque propuesto es estable, reproducible y aplicable en entornos clínicos, y que constituye una base sólida para su validación en fases posteriores, así como para su futura integración en infraestructuras hospitalarias.

En relación con los objetivos del proyecto, se han cumplido tanto el objetivo general como los objetivos específicos planteados, alcanzando los hitos previstos en cada una de las áreas de desarrollo.

Esta segunda fase dentro de la línea de investigación del grupo IASalud supone un salto cualitativo respecto al trabajo previo centrado en el diseño del dispositivo. En esta etapa se ha logrado implementar una arquitectura completa y funcional que integra todas las capas del sistema, consolidando un prototipo capaz de funcionar de manera autónoma y generar información clínicamente relevante. Se ha demostrado la viabilidad técnica del sistema para la digitalización del TRC mediante una arquitectura IoMT escalable y reproducible, sentando las bases para su futura validación e integración clínica. De este modo, el trabajo no solo cumple los objetivos definidos, sino que los supera, proporcionando un marco tecnológico coherente y robusto sobre el que podrán apoyarse las siguientes fases del proyecto.

Desde la perspectiva de la innovación, el proyecto presenta un carácter original, al no existir antecedentes en la literatura científica que apliquen de forma integrada IA embebida, arquitecturas de datos específicas y visualización clínica adaptada a la medición del TRC. Esta combinación abre una línea de investigación en la digitalización inteligente de parámetros fisiológicos, aportando un valor diferencial tanto a nivel técnico como en su aplicabilidad dentro del entorno clínico.

Finalmente, el trabajo se enmarca dentro de un enfoque de sostenibilidad tecnológica y social, al proponer una solución abierta y eficiente que promueve el uso responsable de los recursos en el ámbito sanitario.

8.2 Conclusiones personales

El desarrollo de este proyecto ha sido una experiencia muy valiosa a nivel personal y profesional. Me ha permitido aplicar los conocimientos adquiridos durante el máster en Big Data a un caso de uso real dentro del ámbito biomédico, integrando conceptos de análisis de datos, IA y sistemas embebidos en una línea de investigación que ya formaba parte de mi trayectoria.

A lo largo del trabajo he podido profundizar en áreas técnicas que antes solo conocía de forma teórica, como el procesamiento embebido, la conectividad IoT o el uso de modelos TinyML. Poder llevarlas a la práctica y comprobar su utilidad en un contexto de investigación aplicada ha sido especialmente enriquecedor.

También ha sido una oportunidad para colaborar con un equipo multidisciplinar y participar de forma activa en la evolución de una línea de investigación en la que se combinan innovación tecnológica y utilidad clínica. Este proceso me ha permitido consolidar competencias técnicas, mejorar mi capacidad de análisis y comprender mejor cómo las tecnologías de datos pueden aportar valor real a la monitorización del paciente crítico.

Capítulo 9. FUTURAS LÍNEAS DE TRABAJO

El sistema desarrollado constituye una base sólida sobre la que se pueden plantear diversas líneas de avance, orientadas a su validación clínica, ampliación funcional e integración con infraestructuras hospitalarias reales.

Una de las prioridades a corto plazo es la validación clínica del sistema en entorno real, concretamente en la UCI del Hospital de Moncloa, donde se desarrolla la línea de investigación del grupo IASalud. Esta fase permitirá comparar los resultados del dispositivo con la medición manual del TRC, evaluar su comportamiento en diferentes condiciones fisiológicas y establecer métricas clínicas de precisión, consistencia y utilidad asistencial.

Para ello, se prevé realizar la integración del sistema con el entorno informático hospitalario existente, basado en el sistema *Philips IntelliSpace Critical Care and Anesthesia (ICCA)*. ICCA ofrece interoperabilidad mediante estándares HL7 y ASTM, así como módulos de cálculo, flujo de datos y gestión de dispositivos a través de la pasarela *IntelliBridge*. Esto permitiría incorporar las mediciones del TRC como un nuevo parámetro fisiológico dentro del *flowsheet* de la UCI, junto con el resto de variables monitorizadas. (56)

Para ello, será necesario desarrollar un módulo de comunicación intermedio capaz de estructurar los datos en formato HL7/FHIR y transmitirlos al sistema ICCA, asegurando la sincronización temporal y la trazabilidad de cada medición. Esta integración facilitaría que los valores de TRC, sus etiquetas de calidad y las alertas automáticas generadas por el modelo TinyML quedaran registradas directamente en la historia clínica electrónica del paciente.

Otra línea de trabajo estará centrada en mejorar la arquitectura de datos, incorporando Apache Kafka como capa de mensajería intermedia para aumentar la escalabilidad y la tolerancia a fallos del sistema. Esto permitiría gestionar mayores volúmenes de información y habilitar la comunicación simultánea entre múltiples dispositivos, preparando la infraestructura para entornos clínicos reales con varios pacientes monitorizados en paralelo.

A nivel analítico, se plantea la explotación del Data Lake en formato Parquet mediante herramientas de análisis masivo y ML, con el fin de estudiar patrones temporales del TRC y su relación con otros parámetros clínicos. Esto permitirá crear modelos predictivos más avanzados y adaptar el comportamiento del sistema a las condiciones fisiológicas individuales de cada paciente.

Por último, será necesario realizar una evaluación experimental del rendimiento del modelo embebido, midiendo su latencia, consumo energético y carga computacional bajo diferentes configuraciones del microcontrolador. Estos datos permitirán optimizar el firmware y garantizar un comportamiento estable en entornos de uso intensivo.

Capítulo 10. REFERENCIAS

1. MCGUIRE, Duncan, GOTLIB, Ari and KING, Jordan. Capillary Refill Time. In : *StatPearls*. Online. Treasure Island (FL) : StatPearls Publishing, 2025. [Accessed 2 May 2025]. Available from: http://www.ncbi.nlm.nih.gov/books/NBK557753/NBK557753

2. RAIA, Lisa, GABARRE, Paul, BONNY, Vincent, URBINA, Tomas, MISSRI, Louai, BOELLE, Pierre-Yves, BAUDEL, Jean-Luc, GUIDET, Bertrand, MAURY, Eric, JOFFRE, Jeremie and AIT-OUFELLA, Hafid. Kinetics of capillary refill time after fluid challenge. *Annals of Intensive Care*. 13 August 2022. Vol. 12, no. 1, p. 74. DOI 10.1186/s13613-022-01049-x.

3. LARA, Barbara, ENBERG, Luis, ORTEGA, Marcos, LEON, Paula, KRIPPER, Cristobal, AGUILERA, Pablo, KATTAN, Eduardo, CASTRO, Ricardo, BAKKER, Jan and HERNANDEZ, Glenn. Capillary refill time during fluid resuscitation in patients with sepsis-related hyperlactatemia at the emergency department is related to mortality. *PloS One*. 2017. Vol. 12, no. 11, p. e0188548. DOI 10.1371/journal.pone.0188548.

4. Prueba del llenado capilar ungueal. Online. [Accessed 3 May 2025]. Available from: https://ssl.adam.com/content.aspx?productid=118&pid=5&gid=003394&site=StLukesmedicalcenter.adam.com&login=STLK7926

5. LIMA, Alexandre, JANSEN, Tim C., VAN BOMMEL, Jasper, INCE, Can and BAKKER, Jan. The prognostic value of the subjective assessment of peripheral perfusion in critically ill patients. *Critical Care Medicine*. March 2009. Vol. 37, no. 3, p. 934. DOI 10.1097/CCM.0b013e31819869db.

6. HERNÁNDEZ, Glenn, KATTAN, Eduardo, OSPINA-TASCÓN, Gustavo, BAKKER, Jan, CASTRO, Ricardo, and THE ANDROMEDA-SHOCK STUDY INVESTIGATORS AND THE LATIN AMERICA INTENSIVE CARE NETWORK (LIVEN). Capillary refill time status could identify different clinical phenotypes among septic shock patients fulfilling Sepsis-3 criteria: a post hoc analysis of ANDROMEDA-SHOCK trial. *Intensive Care Medicine*. 1 April 2020. Vol. 46, no. 4, p. 816–818. DOI 10.1007/s00134-020-05960-4.

7. ESPINOZA, Emilio Daniel Valenzuela, WELSH, Sebastián and DUBIN, Arnaldo. Lack of agreement between different observers and methods in the measurement of capillary refill time in healthy volunteers: an observational study. *Revista brasileira de terapia intensiva*. 1 January 2014. Vol. 26, no. 3, p. 269–276.

8. To See or Not to See : A Study on Capillary Refill. Online. [Accessed 2 May 2025]. Available from: https://liu.diva-portal.org/smash/record.jsf?pid=diva2%3A1420852&dswid=3836

9. SHINOZAKI, Koichiro, JACOBSON, Lee S., SAEKI, Kota, KOBAYASHI, Naoki, WEISNER, Steve, FALOTICO, Julianne M., LI, Timmy, KIM, Junhwan, LAMPE, Joshua W. and BECKER, Lance B. The standardized method and clinical experience may improve the reliability of visually

assessed capillary refill time. *The American Journal of Emergency Medicine*. 1 June 2021. Vol. 44, p. 284–290. DOI 10.1016/j.ajem.2020.04.007.

10. SHINOZAKI, Koichiro, CAPILUPI, Michael J., SAEKI, Kota, HIRAHARA, Hideaki, HORIE, Katsuyuki, KOBAYASHI, Naoki, WEISNER, Steve, KIM, Junhwan, LAMPE, Joshua W. and BECKER, Lance B. Low temperature increases capillary blood refill time following mechanical fingertip compression of healthy volunteers: prospective cohort study. *Journal of Clinical Monitoring and Computing*. 1 April 2019. Vol. 33, no. 2, p. 259–267. DOI 10.1007/s10877-018-0159-7.

11. SHAVIT, Itai, BRANT, Rollin, NIJSSEN-JORDAN, Cheri, GALBRAITH, Roger and JOHNSON, David W. A novel imaging technique to measure capillary-refill time: improving diagnostic accuracy for dehydration in young children with gastroenteritis. *Pediatrics*. December 2006. Vol. 118, no. 6, p. 2402–2408. DOI 10.1542/peds.2006-1108.

12. BACHOUR, Raquel P. de Souza, DIAS, Eduardo Lopes and CARDOSO, George C. Skin-color-independent robust assessment of capillary refill time. *Journal of Biophotonics*. 2023. Vol. 16, no. 11, p. e202300063. DOI 10.1002/jbio.202300063.

13. JACQUET-LAGRÈZE, Matthias, SAINT-JEAN, Christophe, BOUËT, Thierry, REYNAUD, Sébastien, RUSTE, Martin and FELLAHI, Jean-Luc. Reliability and reproducibility of the DICART device to assess capillary refill time: a bench and in-silico study. *Journal of Clinical Monitoring and Computing*. 1 October 2023. Vol. 37, no. 5, p. 1409–1412. DOI 10.1007/s10877-023-01027-z.

14. YASUFUMI, Oi, MORIMURA, Naoto, SHIRASAWA, Aya, HONZAWA, Hiroshi, OYAMA, Yutaro, NIIDA, Shoko, ABE, Takeru, IMAKI, Shouhei and TAKEUCHI, Ichiro. Quantitative capillary refill time predicts sepsis in patients with suspected infection in the emergency department: an observational study. *Journal of Intensive Care*. 6 May 2019. Vol. 7, no. 1, p. 29. DOI 10.1186/s40560-019-0382-4.

15. SHERIDAN, David C., CLOUTIER, Robert L., SAMATHAM, Ravi and HANSEN, Matthew L. Point-Of-Care Capillary Refill Technology Improves Accuracy of Peripheral Perfusion Assessment. *Frontiers in Medicine*. 2021. Vol. 8, p. 694241. DOI 10.3389/fmed.2021.694241.

16. KARLEN, Walter, PICKARD, Amelia, DANIELS, Jeremy, KWIZERA, Arthur, IBINGIRA, Charles, DUMONT, Guy and ANSERMINO, J. Mark. Automated Validation of Capillary Refill Time Measurements Using Photo-Plethysmogram from a Portable Device for Effective Triage in Children. In : *2011 IEEE Global Humanitarian Technology Conference*. Online. October 2011. p. 66–71. DOI 10.1109/GHTC.2011.19.

17. BLAXTER, L L, MORRIS, D E, CROWE, J A, HENRY, C, HILL, S, SHARKEY, D, VYAS, H and HAYES-GILL, B R. An automated quasi-continuous capillary refill timing device. *Physiological Measurement*. December 2015. Vol. 37, no. 1, p. 83. DOI 10.1088/0967-3334/37/1/83.

18. MORIMURA, Naoto, TAKAHASHI, Kohei, DOI, Tomoki, OHNUKI, Takahiro, SAKAMOTO, Tetsuya, UCHIDA, Yasuyuki, TAKAHASHI, Hiroki, FUJITA, Takashi and IKEDA, Hiroto. A pilot study

of quantitative capillary refill time to identify high blood lactate levels in critically ill patients. *Emergency medicine journal: EMJ*. June 2015. Vol. 32, no. 6, p. 444–448. DOI 10.1136/emermed-2013-203180.

19. BALLAJI, Hattan K., CORREIA, Ricardo, LIU, Chong, KORPOSH, Serhiy, HAYES-GILL, Barrie R., MUSGROVE, Alison and MORGAN, Stephen P. Optical Fibre Sensor for Capillary Refill Time and Contact Pressure Measurements under the Foot. *Sensors*. January 2021. Vol. 21, no. 18, p. 6072. DOI 10.3390/s21186072.

20. LIU, Chong, CORREIA, Ricardo, BALLAJI, Hattan, KORPOSH, Serhiy, HAYES-GILL, Barrie and MORGAN, Stephen. Optical Fibre Sensor for Simultaneous Measurement of Capillary Refill Time and Contact Pressure. *Sensors*. January 2020. Vol. 20, no. 5, p. 1388. DOI 10.3390/s20051388.

21. ROSLI, Nurdania Irisha and ZAKI, Wan Suhaimizan Wan. Capillary Refill Time Monitoring System using Optical Fibre Sensor. *Evolution in Electrical and Electronic Engineering*. 26 October 2023. Vol. 4, no. 2, p. 433–441. DOI 10.30880/eeee.2023.04.02.053.

22. SHINOZAKI, Masayoshi, SHIMIZU, Rika, SAITO, Daiki, NAKADA, Taka-aki and NAKAGUCHI, Toshiya. Portable measurement device to quantitatively measure capillary refilling time. *Artificial Life and Robotics*. 1 February 2022. Vol. 27, no. 1, p. 48–57. DOI 10.1007/s10015-021-00723-w.

23. SHINOZAKI, Masayoshi, NAKADA, Taka-aki, SAITO, Daiki, TOMITA, Keisuke, NOMURA, Yukihiro and NAKAGUCHI, Toshiya. Cut-Off Value of Capillary Refill Time for Peripheral Circulatory Failure Diagnosis. *Prehospital and Disaster Medicine*. June 2023. Vol. 38, no. 3, p. 319–325. DOI 10.1017/S1049023X23005812.

24. HUNTER, Ryan Brandon, JIANG, Shen, NISHISAKI, Akira, NICKEL, Amanda J., NAPOLITANO, Natalie, SHINOZAKI, Koichiro, LI, Timmy, SAEKI, Kota, BECKER, Lance B., NADKARNI, Vinay M. and MASINO, Aaron J. Supervised Machine Learning Applied to Automate Flash and Prolonged Capillary Refill Detection by Pulse Oximetry. *Frontiers in Physiology*. Online. 6 October 2020. Vol. 11. DOI 10.3389/fphys.2020.564589.

25. LEWANDOWSKA, Katarzyna, WEISBROT, Magdalena, CIELOSZYK, Aleksandra, MĘDRZYCKA-DĄBROWSKA, Wioletta, KRUPA, Sabina and OZGA, Dorota. Impact of Alarm Fatigue on the Work of Nurses in an Intensive Care Environment—A Systematic Review. *International Journal of Environmental Research and Public Health*. January 2020. Vol. 17, no. 22, p. 8409. DOI 10.3390/ijerph17228409.

26. GÜLŞEN, Muaz and ARSLAN, Sevban. The Effect of Alarm Fatigue on the Tendency to Make Medical Errors in Surgical Intensive Care Nurses: A Correlational Study Examining the Role of Moderating Factors. *Healthcare*. January 2025. Vol. 13, no. 6, p. 631. DOI 10.3390/healthcare13060631.

27. COOPER, Robert G. Agile–Stage-Gate Hybrids: The Next Stage for Product DevelopmentBlending Agile and Stage-Gate methods can provide flexibility, speed, and improved communication in new-product development. *Research-Technology Management*. 2 January 2016. Vol. 59, no. 1, p. 21–29. DOI 10.1080/08956308.2016.1117317.

28. GARCÍA-SÁNCHEZ, Enrique Rafael, VARGAS-MARTÍNEZ, Héctor Simón, CANDIA-GARCÍA, Filiberto and CONTRERAS-LIMA, Joel. Agile Stage-Gate Approach for Design, Integration, and Testing of a 1U CubeSat. *Aerospace*. April 2024. Vol. 11, no. 4, p. 324. DOI 10.3390/aerospace11040324.

29. COOPER, Robert G. The Stage-Gate® Product Innovation System: from Idea to Launch. *Wiley Encyclopedia of Management*. 2015. P. 1–13. DOI 10.1002/9781118785317.weom130024.

30. UNITED NATIONS, DEPARTMENT OF ECONOMIC AND SOCIAL AFFAIRS. Transforming our world: the 2030 Agenda for Sustainable Development. Online. [Accessed 15 August 2025]. Available from: https://sdgs.un.org/2030agenda

31. CLARA AIBAR ÁLVAREZ and JOSÉ LUIS LAFUENTE CARRASCO. *Diseño de un dispositivo para la medida del relleno capilar en Cuidados Intensivos*. . Bachelor's thesis. Universidad Europea de Madrid, 2024.

32. HTMLAPP.ALLDATASHEET.COM. TCS3472 datasheet(2/27 Pages) AMSCO. Online. [Accessed 6 June 2025]. Available from: http://htmlapp.alldatasheet.com/html-pdf/560511/AMSCO/TCS3472/306/2/TCS3472.html

33. INC, Interlink Electronics. FSR 402. Online. [Accessed 6 June 2025]. Available from: https://www.interlinkelectronics.com/fsr-402

34. Seeed Studio XIAO, the smallest Arduino boards for tinyML. Online. [Accessed 6 June 2025]. Available from: https://www.seeedstudio.com/xiao-series-page?srsltid=AfmBOorlq9OaHVeXxB9D18uNZ8e2Nwg6DmJs0Bdt9uZNdh2en5pYdy08

35. HUANG, Chenxi, WANG, Jian, WANG, Shuihua and ZHANG, Yudong. Internet of medical things: A systematic review. *Neurocomputing*. 7 November 2023. Vol. 557, p. 126719. DOI 10.1016/j.neucom.2023.126719.

36. NIU, Qinwang, LI, Haoyue, LIU, Yu, QIN, Zhibo, ZHANG, Li-bo, CHEN, Junxin and LYU, Zhihan. Toward the Internet of Medical Things: Architecture, trends and challenges. *Mathematical Biosciences and Engineering*. 2024. Vol. 21, no. 1, p. 650–678. DOI 10.3934/mbe.2024028.

37. MERENDA, Massimo, PORCARO, Carlo and IERO, Demetrio. Edge Machine Learning for AI-Enabled IoT Devices: A Review. *Sensors*. January 2020. Vol. 20, no. 9, p. 2533. DOI 10.3390/s20092533.

38. ANDRIULO, Francesco Cosimo, FIORE, Marco, MONGIELLO, Marina, TRAVERSA, Emanuele and ZIZZO, Vera. Edge Computing and Cloud Computing for Internet of Things: A Review. *Informatics*. December 2024. Vol. 11, no. 4, p. 71. DOI 10.3390/informatics11040071.

39. HEYDARI, Soroush and MAHMOUD, Qusay H. Tiny Machine Learning and On-Device Inference: A Survey of Applications, Challenges, and Future Directions. *Sensors*. January 2025. Vol. 25, no. 10, p. 3191. DOI 10.3390/s25103191.

40. A, Ahila, DAHAN, Fadl, ALROOBAEA, Roobaea, ALGHAMDI, Wael Y., MUSTAFA KHAJA MOHAMMED, HAJJEJ, Fahima, DEEMA MOHAMMED ALSEKAIT and RAAHEMIFAR, Kaamran. A smart IoMT based architecture for E-healthcare patient monitoring system using artificial intelligence algorithms. *Frontiers in Physiology*. Online. 30 January 2023. Vol. 14. DOI 10.3389/fphys.2023.1125952.

41. AWRAHMAN, Banan Jamil, AZIZ FATAH, Chia and HAMAAMIN, Mzhda Yasin. A Review of the Role and Challenges of Big Data in Healthcare Informatics and Analytics. *Computational Intelligence and Neuroscience*. 2022. Vol. 2022, no. 1, p. 5317760. DOI 10.1155/2022/5317760.

42. OSAMA, Manar, ATEYA, Abdelhamied A., SAYED, Mohammed S., HAMMAD, Mohamed, PŁAWIAK, Paweł, ABD EL-LATIF, Ahmed A. and ELSAYED, Rania A. Internet of Medical Things and Healthcare 4.0: Trends, Requirements, Challenges, and Research Directions. *Sensors*. January 2023. Vol. 23, no. 17, p. 7435. DOI 10.3390/s23177435.

43. AL KHATIB, Inas, SHAMAYLEH, Abdulrahim and NDIAYE, Malick. Healthcare and the Internet of Medical Things: Applications, Trends, Key Challenges, and Proposed Resolutions. *Informatics*. September 2024. Vol. 11, no. 3, p. 47. DOI 10.3390/informatics11030047.

44. Wi-Fi - ESP32-C3 - — ESP-IDF Programming Guide latest documentation. Online. [Accessed 10 June 2025]. Available from: https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/api-reference/network/esp_wifi.html

45. ESPRESSIF. arduino-esp32/libraries/WiFi at master · espressif/arduino-esp32. *GitHub*. Online. [Accessed 20 June 2025]. Available from: https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi

46. JONCAS, Roxanne. MQTT 101 – How to Get Started with the lightweight IoT Protocol | The Eclipse Foundation. Online. [Accessed 17 June 2025]. Available from: https://www.eclipse.org/community/eclipse_newsletter/2014/october/article2.php

47. ESPRESSIF. arduino-esp32/libraries/WiFi at master · espressif/arduino-esp32. *GitHub*. Online. [Accessed 20 June 2025]. Available from: https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi

48. GitHub - knolleary/pubsubclient: A client library for the Arduino Ethernet Shield that provides support for MQTT. Online. [Accessed 20 June 2025]. Available from: https://github.com/knolleary/pubsubclient

49. Structured Streaming Programming Guide - Spark 4.0.1 Documentation. Online. [Accessed 21 June 2025]. Available from: https://spark.apache.org/docs/latest/streaming/index.html

50. RESTREPO, Juan P., RIVERA, Juan Carlos, LANIADO, Henry, OSORIO, Pablo and BECERRA, Omar A. Nonparametric Generation of Synthetic Data Using Copulas. *Electronics*. January 2023. Vol. 12, no. 7, p. 1601. DOI 10.3390/electronics12071601.

51. HOUSSOU, Regis, AUGUSTIN, Mihai-Cezar, RAPPOS, Efstratios, BONVIN, Vivien and ROBERT-NICOUD, Stephan. *Generation and Simulation of Synthetic Datasets with Copulas*. Online. 30 March 2022. arXiv. arXiv:2203.17250. arXiv:2203.17250 [cs]

52. BLOCKEEL, Hendrik, DEVOS, Laurens, FRÉNAY, Benoît, NANFACK, Géraldin and NIJSSEN, Siegfried. Decision trees: from efficient prediction to responsible AI. *Frontiers in Artificial Intelligence*. Online. 26 July 2023. Vol. 6. DOI 10.3389/frai.2023.1124553.

53. YAMASHITA, Yui, TAYA, Akihito, WADA, Takumi and TOBE, Yoshito. Decision-tree-based distributed learning for IoT devices. *Journal of Reliable Intelligent Environments*. 21 April 2025. Vol. 11, no. 2, p. 9. DOI 10.1007/s40860-025-00249-z.

54. LATEFA HAMAD AL FRYAN. Processing Decision Tree Data Using Internet of Things (IoT) and Artificial Intelligence Technologies with Special Reference to Medical Application. Online. DOI 10.1155/2022/8626234.

55. SALERNO, Simone. *eloquentarduino/micromlgen*. Online. Python. 7 October 2025. [Accessed 20 June 2025]. Available from: https://github.com/eloquentarduino/micromlgen

56. PHILIPS HEALTHCARE. *IntelliSpace Critical Care and Anesthesia - Instructions for Use* Online. Manual técnico / Instructions for Use. 2024, [no date]. [Accessed 2 August 2025]. Available from: https://www.philips.com/healthcare/product/HCNOCTN123456/intellispace-critical-care-and-anesthesia

ANEXO A. Código Arduino – Escaneo de redes Wi-Fi con la placa Seeed Studio XIAO ESP32-C3

```
#include <WiFi.h>

const char* TARGET_SSID = "";

const char* encTypeStr(wifi_auth_mode_t e) {
  switch (e) {
    case WIFI_AUTH_OPEN: return "OPEN";
    case WIFI_AUTH_WEP: return "WEP";
    case WIFI_AUTH_WPA_PSK: return "WPA";
    case WIFI_AUTH_WPA2_PSK: return "WPA2";
    case WIFI_AUTH_WPA_WPA2_PSK: return "WPA/WPA2";
    case WIFI_AUTH_WPA2_ENTERPRISE: return "WPA2-ENT";
    case WIFI_AUTH_WPA3_PSK: return "WPA3";
    case WIFI_AUTH_WPA2_WPA3_PSK: return "WPA2/WPA3";
    default: return "?";
  }
}

void setup() {
  Serial.begin(115200);
  delay(500);
  Serial.println("\n== Escaneo WiFi (ESP32-C3) ==");
  WiFi.mode(WIFI_STA);
  WiFi.setSleep(false);
  WiFi.disconnect(true, true);
}

void loop() {
  Serial.println("\nBuscando redes... (2-3 s)");
  int n = WiFi.scanNetworks(false, true);
  if (n <= 0) {
    Serial.println("No se encontraron redes.");
  } else {
    Serial.printf("Encontradas %d redes:\n", n);
    Serial.println("-------------------------------------------------
-----------");
    Serial.println("#  SSID                        RSSI  Canal  Banda
Seguridad");
    Serial.println("-------------------------------------------------
-----------");
    for (int i = 0; i < n; i++) {
      String ssid = WiFi.SSID(i);
      int32_t rssi = WiFi.RSSI(i);
      int32_t ch   = WiFi.channel(i);
      const char* band = (ch <= 14) ? "2.4G" : "5G*";
      const char* sec  =
encTypeStr((wifi_auth_mode_t)WiFi.encryptionType(i));
```

```
      bool isTarget = (strlen(TARGET_SSID) > 0 && ssid ==
TARGET_SSID);
      char line[120];
      snprintf(line, sizeof(line), "%-2d %-24s %4ld   %-5ld  %-5s
%s",
               i+1, ssid.c_str(), (long)rssi, (long)ch, band, sec);
      if (isTarget) Serial.print("* "); else Serial.print("  ");
      Serial.println(line);
    }
    Serial.println("(* = tu red objetivo; '5G*' = 5 GHz (no compatible
con ESP32-C3))");
  }
  WiFi.scanDelete();
  delay(5000);
}
```

ANEXO B. Código de Arduino – Cliente MQTT básico en la placa Seeed Studio XIAO ESP32-C3

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* WIFI_SSID = "NOMBRE_WIFI";
const char* WIFI_PASS = "CONTRASEÑA_WIFI";

const char* MQTT_HOST = "test.mosquitto.org";
const uint16_t MQTT_PORT = 1883;

const char* BASE = "tfm_clara_demo";
String pubTopic = String(BASE) + "/status";

WiFiClient net;
PubSubClient mqtt(net);

bool ensureMqtt() {
  if (mqtt.connected()) return true;
  mqtt.setServer(MQTT_HOST, MQTT_PORT);
  String clientId = "ESP32C3-" +
String((uint32_t)ESP.getEfuseMac(), HEX);
  return mqtt.connect(clientId.c_str());
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.setSleep(false);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(300);
    Serial.print(".");
  }
  Serial.println("\nWiFi OK, IP: " + WiFi.localIP().toString());
}

void loop() {
  if (!ensureMqtt()) {
    delay(1000);
    return;
  }
  mqtt.loop();

  static unsigned long last = 0;
  if (millis() - last > 5000) {
    last = millis();
    String payload = String("{\"ip\":\"") +
WiFi.localIP().toString() +
                     "\",\"rssi\":" + String(WiFi.RSSI()) + "}";
```

```
    bool ok = mqtt.publish(pubTopic.c_str(), payload.c_str());
    Serial.println(ok ? "[MQTT] status OK" : "[MQTT] status
FAIL");
  }
}
```

ANEXO C: Código de Arduino – Cliente MQTT avanzado (reconexión, LWT, comandos y telemetría) en Seeed Studio XIAO ESP32-C3

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* WIFI_SSID = "NOMBRE_WIFI";
const char* WIFI_PASS = "CONTRASEÑA_WIFI";

const char* MQTT_HOST = "test.mosquitto.org";
const uint16_t MQTT_PORT = 1883;

const char* BASE = "tfm_clara_demo";
String T_STATUS = String(BASE) + "/status";
String T_CMD    = String(BASE) + "/cmd";
String T_ACK    = String(BASE) + "/ack";
String T_LWT    = String(BASE) + "/lwt";

WiFiClient net;
PubSubClient mqtt(net);

unsigned long lastPub = 0;
uint32_t tick = 0;

void printlnTitled(const char* title, const String& v) {
  Serial.print(title);
  Serial.println(v);
}

void mqttCallback(char* topic, byte* payload, unsigned int len) {
  String msg;
  for (unsigned int i = 0; i < len; i++) msg += (char)payload[i];
  Serial.print("\n[MQTT] RX ");
  Serial.print(topic);
  Serial.print(" : ");
  Serial.println(msg);

  String s = msg;
  s.trim();
  s.toUpperCase();

  if (s == "LED_ON") {
    digitalWrite(LED_BUILTIN, HIGH);
    mqtt.publish(T_ACK.c_str(), "{\"ack\":\"LED_ON\"}");
  } else if (s == "LED_OFF") {
    digitalWrite(LED_BUILTIN, LOW);
    mqtt.publish(T_ACK.c_str(), "{\"ack\":\"LED_OFF\"}");
  } else if (s == "PING") {
```

```
    String ack = String("{\"ack\":\"PONG\",\"ms\":") +
String(millis()) + "}";
    mqtt.publish(T_ACK.c_str(), ack.c_str());
  } else {
    String ack = String("{\"ack\":\"UNKNOWN\",\"cmd\":\"") + msg +
"\"}";
    mqtt.publish(T_ACK.c_str(), ack.c_str());
  }
}

bool mqttConnect() {
  if (mqtt.connected()) return true;

  String clientId = "ESP32C3-" +
String((uint32_t)ESP.getEfuseMac(), HEX);
  mqtt.setServer(MQTT_HOST, MQTT_PORT);
  mqtt.setCallback(mqttCallback);
  Serial.print("[MQTT] Conectando... ");
  mqtt.setKeepAlive(5);

  bool ok = mqtt.connect(
    clientId.c_str(),
    nullptr, nullptr,
    T_LWT.c_str(),
    0, false,
    "offline",
    true
  );

  if (!ok) {
    Serial.print("FAIL rc=");
    Serial.println(mqtt.state());
    return false;
  }

  Serial.println("OK");
  mqtt.publish(T_LWT.c_str(), "online", true);
  mqtt.subscribe(T_CMD.c_str());
  printlnTitled("[MQTT] Subscrito a ", T_CMD);
  return true;
}

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);

  Serial.begin(115200);
  delay(300);
  Serial.println("\n== TFM MQTT (broker público) ==");

  WiFi.mode(WIFI_STA);
  WiFi.setSleep(false);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
```

```
  Serial.print("WiFi conectando");
  unsigned long t0 = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - t0 < 20000) {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("WiFi no conecta");
    return;
  }
  Serial.print("WiFi OK. IP: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  if (WiFi.status() != WL_CONNECTED) return;

  if (!mqttConnect()) {
    delay(1500);
    return;
  }
  mqtt.loop();

  if (millis() - lastPub > 5000) {
    lastPub = millis();
    tick++;

    int raw = analogRead(A0);

    String payload = String("{\"ip\":\"") +
WiFi.localIP().toString() +
                     "\",\"rssi\":" + String(WiFi.RSSI()) +
                     ",\"tick\":" + String(tick) +
                     ",\"a0\":" + String(raw) + "}";
    bool ok = mqtt.publish(T_STATUS.c_str(), payload.c_str(),
true);
    Serial.print("[MQTT] pub ");
    Serial.print(T_STATUS);
    Serial.print(" -> ");
    Serial.println(ok ? "OK" : "FAIL");
  }
}
```

ANEXO D: Script PowerShell – Medición de latencia MQTT (RTT PING/ACK)

```
$pub      = "C:\Program Files\mosquitto\mosquitto_pub.exe"
$sub      = "C:\Program Files\mosquitto\mosquitto_sub.exe"
$broker   = "test.mosquitto.org"
$topicCmd = "tfm_clara_demo/cmd"
$topicAck = "tfm_clara_demo/ack"

$results = @()

1..10 | ForEach-Object {
  $tmp = [System.IO.Path]::GetTempFileName()
  $p = Start-Process -FilePath $sub -ArgumentList "-h $broker -t
$topicAck -C 1" `
         -NoNewWindow -PassThru -RedirectStandardOutput $tmp
  Start-Sleep -Milliseconds 200

  $t0 = Get-Date
  & $pub -h $broker -t $topicCmd -m "PING" | Out-Null
  Wait-Process $p.Id
  $dt = [math]::Round(((Get-Date) - $t0).TotalMilliseconds, 1)
  $results += $dt

  Get-Content $tmp
  Remove-Item $tmp -Force
}

"RTTs: $results"
$min = ($results | Measure-Object -Minimum).Minimum
$avg = [math]::Round(($results | Measure-Object -Average).Average,
1)
$max = ($results | Measure-Object -Maximum).Maximum
"min/avg/max = $min / $avg / $max ms"
```

ANEXO E. Definición de tabla SQL para almacenamiento de observaciones de TRC

```
CREATE TABLE IF NOT EXISTS public.trc_observation (
 ts           timestamptz NOT NULL,
 device_id    text        NOT NULL,
 patient_id   text,
 rssi         int,
 trc_ms       int,
 qa           jsonb,
 env          jsonb,
 raw_payload  jsonb       NOT NULL,
 ingested_at  timestamptz NOT NULL DEFAULT now()
);
```

ANEXO F. Script PySpark – Ingesta de archivos JSON en la base de datos PostgreSQL

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType,
IntegerType
from pyspark.sql.functions import to_timestamp, current_timestamp,
to_json, struct, col

INCOMING_DIR = r"/data/incoming"

spark = (
    SparkSession.builder
        .appName("tfm-ingest-files-once")
        .getOrCreate()
)

props = {
    "user": "usuario",
    "password": "contraseña",
    "driver": "org.postgresql.Driver",
    "stringtype": "unspecified"
}

schema = StructType([
    StructField("ts",         StringType(),  True),
    StructField("device_id", StringType(),  True),
    StructField("patient_id",StringType(),  True),
    StructField("rssi",       IntegerType(), True),
    StructField("trc_ms",     IntegerType(), True),
    StructField("qa",         StringType(),  True),
    StructField("env",        StringType(),  True),
])

df = (
    spark.read
         .schema(schema)
         .json(INCOMING_DIR)
)

df = df.withColumn("ts", to_timestamp(col("ts"), "yyyy-MM-
dd'T'HH:mm:ssX"))

df = (
    df.withColumn("raw_payload", to_json(struct([col(c) for c in
df.columns])))
      .withColumn("ingested_at", current_timestamp())
)

jdbc_url = "jdbc:postgresql://localhost:5432/tfm"
```

```
out = df.select("ts","device_id","patient_id","rssi","trc_ms",
                "qa","env","raw_payload","ingested_at")

(out.write
   .mode("append")
   .jdbc(jdbc_url, "public.trc_observation", properties=props))

spark.stop()
```

ANEXO G. Ejecución del job PySpark mediante spark-submit en PowerShell

```
$py = "C:\Python310\python.exe"

spark-submit `
  --conf spark.pyspark.python="$py" `
  --conf spark.pyspark.driver.python="$py" `
  --packages org.postgresql:postgresql:42.7.3 `
  C:\spark_jobs\ingest_files_once.py
```

ANEXO H. Consultas SQL de verificación de datos en la base de datos PostgreSQL-- 1) Conteo rápido

```sql
SELECT count(*)
FROM public.trc_observation;

SELECT
  (raw_payload::jsonb ->> 'device_id')    AS device_id,
  (raw_payload::jsonb ->> 'patient_id')   AS patient_id,
  (raw_payload::jsonb ->> 'rssi')::int    AS rssi,
  (raw_payload::jsonb ->> 'trc_ms')::int AS trc_ms,
  ts,
  ingested_at
FROM public.trc_observation
ORDER BY ingested_at DESC
LIMIT 10;
```

ANEXO I. Script Python – Generación de archivos JSON de prueba para la ingesta de datosimport os, time, json

```
import os
import time
import json
from datetime import datetime, timezone

OUT_DIR = r"/data/incoming_streaming"

records = [
    {
        "ts": datetime.now(timezone.utc).strftime("%Y-%m-
%dT%H:%M:%SZ"),
        "device_id": "esp32_demo_01",
        "patient_id": "anon-001",
        "rssi": -58,
        "trc_ms": 2300,
        "qa": "{\"saturation\": false}",
        "env": "{\"temp_c\": 22.5}"
    },
    {
        "ts": datetime.now(timezone.utc).strftime("%Y-%m-
%dT%H:%M:%SZ"),
        "device_id": "esp32_demo_01",
        "patient_id": "anon-001",
        "rssi": -60,
        "trc_ms": 2400,
        "qa": "{\"saturation\": false}",
        "env": "{\"temp_c\": 22.6}"
    }
]

os.makedirs(OUT_DIR, exist_ok=True)
fname = f"batch_{int(time.time())}.json"
fpath = os.path.join(OUT_DIR, fname)

with open(fpath, "w", encoding="utf-8") as f:
    for rec in records:
        f.write(json.dumps(rec, ensure_ascii=False))
        f.write("\n")

print("Archivo generado:", fpath)
```

ANEXO J. Script PySpark – Ingesta en streaming de JSON hacia PostgreSQL

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType,
IntegerType
from pyspark.sql.functions import to_timestamp, current_timestamp,
to_json, struct, col

INCOMING_DIR    = r"/data/incoming_streaming"
CHECKPOINT_DIR = r"/data/checkpoint_streaming"

spark = (
    SparkSession.builder
        .appName("tfm-ingest-streaming")
        .getOrCreate()
)

schema = StructType([
    StructField("ts", StringType(), True),
    StructField("device_id", StringType(), True),
    StructField("patient_id", StringType(), True),
    StructField("rssi", IntegerType(), True),
    StructField("trc_ms", IntegerType(), True),
    StructField("qa", StringType(), True),
    StructField("env", StringType(), True),
])

df = (
    spark.readStream
         .schema(schema)
         .json(INCOMING_DIR)
)

df = (
    df.withColumn("ts", to_timestamp(col("ts"), "yyyy-MM-
dd'T'HH:mm:ssX"))
      .withColumn("raw_payload", to_json(struct([col(c) for c in
df.columns])))
      .withColumn("ingested_at", current_timestamp())
      .dropDuplicates(["ts", "device_id", "trc_ms"])
)

jdbc_url = "jdbc:postgresql://localhost:5432/tfm"
props = {
    "user": "usuario",
    "password": "contraseña",
    "driver": "org.postgresql.Driver"
}

def write_batch(batch_df, batch_id: int):
```

```
        n = batch_df.count()
        if n == 0:
            return
        (batch_df

    .select("ts","device_id","patient_id","rssi","trc_ms","qa","env","
    raw_payload","ingested_at")
            .write
            .mode("append")
            .jdbc(jdbc_url, "public.trc_observation",
    properties=props)
        )

    query = (
        df.writeStream
          .outputMode("append")
          .option("checkpointLocation", CHECKPOINT_DIR)
          .trigger(processingTime="5 seconds")
          .foreachBatch(write_batch)
          .start()
    )

    query.awaitTermination()
```

ANEXO K. Ejecución del job PySpark de ingesta en streaming mediante spark-submit en PowerShell

```
$py = "C:\Python310\python.exe"

spark-submit `
  --conf spark.pyspark.python="$py" `
  --conf spark.pyspark.driver.python="$py" `
  --packages org.postgresql:postgresql:42.7.3 `
  C:\spark_jobs\ingest_streaming.py
```

ANEXO L. Consultas SQL de verificación de registros en DBeaver

```sql
SELECT COUNT(*)
FROM public.trc_observation;

SELECT
  (raw_payload::jsonb ->> 'device_id')    AS device_id,
  (raw_payload::jsonb ->> 'patient_id')   AS patient_id,
  (raw_payload::jsonb ->> 'rssi')::int    AS rssi,
  (raw_payload::jsonb ->> 'trc_ms')::int AS trc_ms,
  ts,
  ingested_at
FROM public.trc_observation
ORDER BY ingested_at DESC
LIMIT 10;
```

ANEXO M. Consultas SQL avanzadas de verificación de integridad y latencia de datos

```sql
SELECT ts, device_id, trc_ms, COUNT(*) AS c
FROM public.trc_observation
GROUP BY 1, 2, 3
HAVING COUNT(*) > 1
ORDER BY c DESC;

SELECT md5(raw_payload::text) AS payload_hash, COUNT(*) AS c
FROM public.trc_observation
GROUP BY 1
HAVING COUNT(*) > 1
ORDER BY c DESC;

SELECT
  NOW() - MAX(ingested_at) AS lag_aproximado
FROM public.trc_observation;
```

ANEXO N. Código Arduino – Publicación MQTT con esquema canónico en la placa Seeed Studio XIAO ESP32-C3

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <time.h>

const char* WIFI_SSID = "SSID";
const char* WIFI_PASS = "PASSWORD";

const char* MQTT_HOST = "test.mosquitto.org";
const uint16_t MQTT_PORT = 1883;

const char* BASE = "tfm_clara_demo";
const char* DEVICE_ID  = "esp32_demo_01";
const char* PATIENT_ID = "anon_001";

const char* NTP1 = "pool.ntp.org";
const char* NTP2 = "time.nist.gov";

String T_STATUS = String(BASE) + "/status";
String T_CMD    = String(BASE) + "/cmd";
String T_ACK    = String(BASE) + "/ack";
String T_LWT    = String(BASE) + "/lwt";

#ifndef LED_BUILTIN
#define LED_BUILTIN 2
#endif

WiFiClient net;
PubSubClient mqtt(net);

unsigned long lastPub = 0;
uint32_t tick = 0;

String iso8601_utc() {
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo, 5000)) {
    return "1970-01-01T00:00:00Z";
  }
  char buf[25];
  strftime(buf, sizeof(buf), "%Y-%m-%dT%H:%M:%SZ", &timeinfo);
  return String(buf);
}

void mqttCallback(char* topic, byte* payload, unsigned int len) {
  String msg;
  for (unsigned int i = 0; i < len; i++) msg += (char)payload[i];
  String s = msg; s.trim(); s.toUpperCase();
```

```
  if (s == "LED_ON") {
    digitalWrite(LED_BUILTIN, HIGH);
    mqtt.publish(T_ACK.c_str(), "{\"ack\":\"LED_ON\"}");
  } else if (s == "LED_OFF") {
    digitalWrite(LED_BUILTIN, LOW);
    mqtt.publish(T_ACK.c_str(), "{\"ack\":\"LED_OFF\"}");
  } else if (s == "PING") {
    String ack = String("{\"ack\":\"PONG\",\"ms\":") +
String(millis()) + "}";
    mqtt.publish(T_ACK.c_str(), ack.c_str());
  } else {
    String ack = String("{\"ack\":\"UNKNOWN\",\"cmd\":\"") + msg +
"\"}";
    mqtt.publish(T_ACK.c_str(), ack.c_str());
  }
}

bool mqttConnect() {
  if (mqtt.connected()) return true;
  mqtt.setServer(MQTT_HOST, MQTT_PORT);
  mqtt.setCallback(mqttCallback);
  mqtt.setKeepAlive(5);
  String clientId = "ESP32C3-" +
String((uint32_t)ESP.getEfuseMac(), HEX);
  bool ok = mqtt.connect(clientId.c_str(), nullptr, nullptr,
T_LWT.c_str(), 0, true, "offline");
  if (!ok) return false;
  mqtt.publish(T_LWT.c_str(), "online", true);
  mqtt.subscribe(T_CMD.c_str());
  return true;
}

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
  Serial.begin(115200);
  delay(300);
  WiFi.mode(WIFI_STA);
  WiFi.setSleep(false);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  unsigned long t0 = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - t0 < 20000) {
    delay(500);
  }
  if (WiFi.status() != WL_CONNECTED) return;
  configTime(0, 0, NTP1, NTP2);
  struct tm ti;
  int tries = 0;
  while (!getLocalTime(&ti) && tries++ < 20) { delay(500); }
}

void loop() {
  if (WiFi.status() != WL_CONNECTED) return;
```

```
  if (!mqttConnect()) { delay(1500); return; }
  mqtt.loop();
  if (millis() - lastPub > 5000) {
    lastPub = millis();
    tick++;
    int raw = analogRead(A0);
    int trc_ms = millis() % 5000;
    String payload =
      String("{")
      + "\"v\":1,"
      + "\"ts\":\"" + iso8601_utc() + "\","
      + "\"device_id\":\"" + String(DEVICE_ID) + "\","
      + "\"patient_id\":\"" + String(PATIENT_ID) + "\","
      + "\"rssi\":" + String(WiFi.RSSI()) + ","
      + "\"trc_ms\":" + String(trc_ms) + ","
      + "\"qa\":\"{\\\"saturation\\\":false}\","
      + "\"env\":\"{\\\"temp_c\\\":22.5}\","
      + "\"data\":{"
        "\"ip\":\"" + WiFi.localIP().toString() + "\","
        "\"tick\":" + String(tick) + ","
        "\"a0\":" + String(raw) +
      "}"
      "}";
    mqtt.publish(T_STATUS.c_str(), payload.c_str(), true);
  }
}
```

ANEXO O. Script Python – Bridge MQTT a archivos JSON para la ingesta de datos

```
import os
import time
import json
import signal
from typing import Optional
import paho.mqtt.client as mqtt

MQTT_HOST = "test.mosquitto.org"
MQTT_PORT = 1883
MQTT_TOPIC = "tfm_clara_demo/status"

OUT_DIR = r"/data/incoming_streaming"

ROTATE_EVERY_MSGS: Optional[int] = None
ROTATE_EVERY_SECONDS: Optional[int] = 15
MAX_LINES_PER_FILE: Optional[int] = None

current_fp = None
current_path = None
opened_at = None
msg_count_in_file = 0
total_msgs = 0
running = True

def _ensure_outdir():
    os.makedirs(OUT_DIR, exist_ok=True)

def _ts_filename():
    ts = int(time.time())
    return os.path.join(OUT_DIR, f"mqtt_batch_{ts}.json")

def _open_new_file():
    global current_fp, current_path, opened_at, msg_count_in_file
    current_path = _ts_filename()
    current_fp = open(current_path, "a", encoding="utf-8")
    opened_at = time.time()
    msg_count_in_file = 0

def _close_file():
    global current_fp
    if current_fp is not None:
        try:
            current_fp.flush()
            current_fp.close()
        finally:
            current_fp = None

def _should_rotate() -> bool:
```

```
    if current_fp is None:
        return True
    if ROTATE_EVERY_MSGS is not None and msg_count_in_file >=
ROTATE_EVERY_MSGS:
        return True
    if ROTATE_EVERY_SECONDS is not None and (time.time() -
opened_at) >= ROTATE_EVERY_SECONDS:
        return True
    if MAX_LINES_PER_FILE is not None and msg_count_in_file >=
MAX_LINES_PER_FILE:
        return True
    return False

def _graceful_exit(*_):
    global running
    running = False

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        client.subscribe(MQTT_TOPIC)

def on_message(client, userdata, msg):
    global current_fp, msg_count_in_file, total_msgs
    try:
        text = msg.payload.decode("utf-8", errors="strict")
        obj = json.loads(text)
        line = json.dumps(obj, ensure_ascii=False)
        if _should_rotate():
            _close_file()
            _open_new_file()
        current_fp.write(line + "\n")
        current_fp.flush()
        msg_count_in_file += 1
        total_msgs += 1
    except Exception:
        pass

def main():
    _ensure_outdir()
    signal.signal(signal.SIGINT, _graceful_exit)
    try:
        signal.signal(signal.SIGTERM, _graceful_exit)
    except Exception:
        pass
    _open_new_file()
    client = mqtt.Client(client_id="", clean_session=True,
userdata=None, protocol=mqtt.MQTTv311)
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect(MQTT_HOST, MQTT_PORT, keepalive=30)
    client.loop_start()
    try:
        while running:
```

```
                time.sleep(0.2)
        finally:
            client.loop_stop()
            client.disconnect()
            _close_file()

if __name__ == "__main__":
    main()
```

ANEXO P. Código Arduino – Firmware ESP32-C3: Publicación MQTT con “payload clínico” (status y crt)

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include "Adafruit_TCS34725.h"
#include <time.h>

const char* WIFI_SSID = "SSID";
const char* WIFI_PASS = "PASSWORD";

const char* MQTT_HOST = "test.mosquitto.org";
const uint16_t MQTT_PORT = 1883;

const char* DEVICE_ID  = "esp32_demo_01";
const char* PATIENT_ID = "anon_001";

String T_BASE   = String("tfm_clara_demo/") + DEVICE_ID + "/";
String T_STATUS = T_BASE + "status";
String T_CRT    = T_BASE + "crt";
String T_CMD    = T_BASE + "cmd";
String T_ACK    = T_BASE + "ack";
String T_LWT    = T_BASE + "lwt";

#ifndef LED_BUILTIN
#define LED_BUILTIN 2
#endif

// ---- Sensores ----
const int FSR_PIN = A0;
Adafruit_TCS34725 tcs(TCS34725_INTEGRATIONTIME_24MS,
TCS34725_GAIN_4X);

// ---- Estados CRT ----
enum Phase { IDLE, COMPRESS, HOLD, RELEASED, RECOVERY };
Phase phase = IDLE;

// ---- Parámetros de evento/QA ----
const float RECOVERY_THRESH_PCT = 0.90f;
const int   HOLD_TARGET_MS = 1000;
const int   HOLD_TOL_MS    = 150;
const int   MIN_AMBIENT_LUX = 50;

// FSR umbrales (mV)
const int FSR_PRESS_MV   = 800;
const int FSR_RELEASE_MV = 400;
```

```
// Sampling y buffers de traza
const int   SAMPLE_PERIOD_MS = 50;
const size_t TRACE_MAX = 64;
int trace_ts[TRACE_MAX];
float trace_force[TRACE_MAX];
float trace_color[TRACE_MAX];
size_t trace_len = 0;

// Variables runtime
WiFiClient esp;
PubSubClient mqtt(esp);

unsigned long lastStatusMs = 0;
unsigned long lastSampleMs = 0;
unsigned long tCompressStart = 0;
unsigned long tCompressEnd = 0;
unsigned long tRelease = 0;

float ema_color = 0.0f;
bool ema_init = false;

int last_fsr_mv = 0;
int peak_fsr_mv = 0;
int ambient_lux = 0;
int sqi = 0;

// ---- Utilidades ----
String iso8601_utc_from(time_t t) {
  char buf[25];
  struct tm tm;
  gmtime_r(&t, &tm);
  strftime(buf, sizeof(buf), "%Y-%m-%dT%H:%M:%SZ", &tm);
  return String(buf);
}

bool waitForTime(uint16_t timeout_ms = 12000) {
  uint32_t t0 = millis();
  while ((time(nullptr) < 1700000000) && (millis() - t0 <
timeout_ms)) {
    delay(200);
  }
  return (time(nullptr) >= 1700000000);
}

bool readColor(float &color_idx, int &lux_out) {
  uint16_t r,g,b,c;
  if (!tcs.getRawData(&r,&g,&b,&c)) return false;
  lux_out = (int)tcs.calculateLux(r,g,b);
  float denom = (c > 0 ? c : 1);
  color_idx = (float)g / denom;
  return true;
}
```

```
int fsrMilliVolts() {
  int raw = analogRead(FSR_PIN);
  return map(raw, 0, 4095, 0, 3300);
}

float mvToNewtonApprox(int mv) {
  return (float)mv / 80.0f;
}

void appendTrace(unsigned long t0_ms, unsigned long now_ms, int
fsr_mv, float color_idx) {
  if (trace_len >= TRACE_MAX) return;
  trace_ts[trace_len] = (int)(now_ms - t0_ms);
  trace_force[trace_len] = mvToNewtonApprox(fsr_mv);
  trace_color[trace_len] = color_idx;
  trace_len++;
}

void mqttCallback(char* topic, byte* payload, unsigned int len) {
  String msg; msg.reserve(len);
  for (unsigned int i=0;i<len;i++) msg += (char)payload[i];
  String s = msg; s.trim(); s.toUpperCase();
  if (s == "LED_ON") { digitalWrite(LED_BUILTIN, HIGH);
mqtt.publish(T_ACK.c_str(), "{\"ack\":\"LED_ON\"}"); }
  else if (s == "LED_OFF") { digitalWrite(LED_BUILTIN, LOW);
mqtt.publish(T_ACK.c_str(), "{\"ack\":\"LED_OFF\"}"); }
  else if (s == "PING") { String ack =
String("{\"ack\":\"PONG\",\"ms\":") + String(millis()) + "}";
mqtt.publish(T_ACK.c_str(), ack.c_str()); }
  else { String ack = String("{\"ack\":\"UNKNOWN\",\"cmd\":\"") +
msg + "\"}"; mqtt.publish(T_ACK.c_str(), ack.c_str()); }
}

bool mqttConnect() {
  if (mqtt.connected()) return true;
  mqtt.setServer(MQTT_HOST, MQTT_PORT);
  mqtt.setBufferSize(4096);
  mqtt.setCallback(mqttCallback);
  mqtt.setKeepAlive(5);
  String clientId = "ESP32C3-" +
String((uint32_t)ESP.getEfuseMac(), HEX);
  bool ok = mqtt.connect(clientId.c_str(), nullptr, nullptr,
T_LWT.c_str(), 0, true, "{\"online\":false}");
  if (!ok) return false;
  mqtt.publish(T_LWT.c_str(), "{\"online\":true}", true);
  mqtt.subscribe(T_CMD.c_str());
  return true;
}

// ---- Publicaciones ----
void publishStatus() {
  time_t nowEpoch = time(nullptr);
  DynamicJsonDocument doc(256);
```

```
  doc["v"] = 1;
  doc["evt_type"] = "status";
  doc["ts"] = iso8601_utc_from(nowEpoch);
  doc["device_id"] = DEVICE_ID;
  doc["patient_id"] = PATIENT_ID;
  doc["rssi"] = WiFi.RSSI();
  JsonObject data = doc.createNestedObject("data");
  data["ip"] = WiFi.localIP().toString();
  char out[256]; size_t n = serializeJson(doc, out, sizeof(out));
  mqtt.publish(T_STATUS.c_str(), out, n);
}

void publishCRT(unsigned long crt_ms, float color_baseline, float
color_min, float color_recovery_pct,
                int ambient_lux_value, float force_peak_n, int
force_hold_ms_value, int sqi_value,
                unsigned long t0, unsigned long t1, unsigned long
t2) {
  time_t nowEpoch = time(nullptr);

  DynamicJsonDocument trace(1024);
  JsonArray ts_ms = trace.createNestedArray("ts_ms");
  JsonArray force = trace.createNestedArray("force");
  JsonArray color = trace.createNestedArray("color");
  for (size_t i=0;i<trace_len;i++) {
    ts_ms.add(trace_ts[i]);
    force.add(trace_force[i]);
    color.add(trace_color[i]);
  }

  String qa_str = String("{")
    + "\"force_in_range\":" + String((force_peak_n >= 5.0f &&
force_peak_n <= 8.0f) ? "true" : "false") + ","
    + "\"hold_in_range\":"  + String((abs(force_hold_ms_value -
HOLD_TARGET_MS) <= HOLD_TOL_MS) ? "true" : "false") + ","
    + "\"motion_high\":false,"
    + "\"light_low\":"      + String((ambient_lux_value <
MIN_AMBIENT_LUX) ? "true" : "false")
    + "}";

  DynamicJsonDocument doc(4096);
  doc["v"] = 1;
  doc["evt_type"] = "crt";
  doc["evt_id"] = String("crt-") + String(random(100000,999999));
  doc["ts"] = iso8601_utc_from(nowEpoch);
  doc["device_id"] = DEVICE_ID;
  doc["patient_id"] = PATIENT_ID;
  doc["t_compress_start"] = iso8601_utc_from(nowEpoch - ((millis()
- t0)/1000));
  doc["t_compress_end"]   = iso8601_utc_from(nowEpoch - ((millis()
- t1)/1000));
  doc["t_recovery_end"]   = iso8601_utc_from(nowEpoch);
  doc["crt_ms"] = (int)crt_ms;
```

```
  doc["recovery_thresh"] = RECOVERY_THRESH_PCT;
  doc["color_baseline"] = color_baseline;
  doc["color_min"] = color_min;
  doc["color_recovery_pct"] = color_recovery_pct;
  doc["ambient_lux"] = ambient_lux_value;
  doc["force_peak_n"] = force_peak_n;
  doc["force_hold_ms"] = force_hold_ms_value;
  doc["sqi"] = sqi_value;
  doc["qa"] = qa_str;
  doc["trace"] = trace.as<JsonObject>();

  char out[4096]; size_t n = serializeJson(doc, out, sizeof(out));
  mqtt.publish(T_CRT.c_str(), out, n);
}

// ---- Setup/Loop ----
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);

  Serial.begin(115200);
  delay(200);

  Wire.begin();
  tcs.begin();

  WiFi.mode(WIFI_STA);
  WiFi.setSleep(false);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  unsigned long t0 = millis();
  while (WiFi.status() != WL_CONNECTED && millis()-t0 < 20000) {
delay(300); }
  if (WiFi.status() != WL_CONNECTED) return;

  configTzTime("UTC0", "pool.ntp.org", "time.nist.gov",
"time.google.com");
  waitForTime(15000);
}

void loop() {
  if (WiFi.status() != WL_CONNECTED) { delay(200); return; }
  if (!mqttConnect()) { delay(500); return; }
  mqtt.loop();

  unsigned long nowMs = millis();

  if (nowMs - lastStatusMs > 10000) {
    lastStatusMs = nowMs;
    publishStatus();
  }

  if (nowMs - lastSampleMs < SAMPLE_PERIOD_MS) return;
  lastSampleMs = nowMs;
```

```
  float color_idx;
  if (!readColor(color_idx, ambient_lux)) return;
  int fsr_mv = fsrMilliVolts();

  if (!ema_init) { ema_color = color_idx; ema_init = true; }
  ema_color = 0.90f * ema_color + 0.10f * color_idx;

  switch (phase) {
    case IDLE:
      peak_fsr_mv = 0; trace_len = 0; sqi = 95;
      if (fsr_mv >= FSR_PRESS_MV) { phase = COMPRESS;
tCompressStart = nowMs; appendTrace(tCompressStart, nowMs, fsr_mv,
color_idx); peak_fsr_mv = fsr_mv; }
      break;

    case COMPRESS:
      appendTrace(tCompressStart, nowMs, fsr_mv, color_idx);
      if (fsr_mv > peak_fsr_mv) peak_fsr_mv = fsr_mv;
      if (fsr_mv >= FSR_PRESS_MV && (nowMs - tCompressStart) >=
100) { phase = HOLD; }
      break;

    case HOLD:
      appendTrace(tCompressStart, nowMs, fsr_mv, color_idx);
      if ((nowMs - tCompressStart) >= HOLD_TARGET_MS) {
tCompressEnd = nowMs; phase = RELEASED; }
      if (fsr_mv < FSR_RELEASE_MV) { tCompressEnd = nowMs; phase =
RELEASED; }
      break;

    case RELEASED:
      appendTrace(tCompressStart, nowMs, fsr_mv, color_idx);
      if (fsr_mv < FSR_RELEASE_MV) { tRelease = nowMs; phase =
RECOVERY; }
      break;

    case RECOVERY:
      appendTrace(tCompressStart, nowMs, fsr_mv, color_idx);
      if (color_idx >= RECOVERY_THRESH_PCT * ema_color || (nowMs -
tRelease) > 5000) {
        unsigned long crt_ms = (nowMs - tRelease);
        float color_baseline = ema_color;
        float color_min = 1.0f;
        for (size_t i=0;i<trace_len;i++) { if (trace_color[i] <
color_min) color_min = trace_color[i]; }
        float color_recovery_pct = (color_baseline > 0.0f) ?
(trace_color[trace_len-1] / color_baseline) : 0.0f;
        int force_hold_ms_value = (int)(tCompressEnd -
tCompressStart);
        float force_peak_n = mvToNewtonApprox(peak_fsr_mv);
```

```
            publishCRT(crt_ms, color_baseline, color_min,
    color_recovery_pct, ambient_lux, force_peak_n,
    force_hold_ms_value, sqi, tCompressStart, tCompressEnd, nowMs);
            phase = IDLE;
          }
          break;
      }
    }
```

ANEXO Q. Script Python – Bridge MQTT → NDJSON con rotación por lotes

```
import os
import time
import json
import signal
from datetime import datetime, timezone
from typing import Optional
import paho.mqtt.client as mqtt

MQTT_HOST = "test.mosquitto.org"
MQTT_PORT = 1883
MQTT_TOPIC = "tfm_clara_demo/#"
OUT_DIR = r"/data/incoming_streaming"

ROTATE_EVERY_MSGS: Optional[int] = 1
ROTATE_EVERY_SECONDS: Optional[int] = None
MAX_LINES_PER_FILE: Optional[int] = None

current_fp = None
current_path = None
opened_at = 0.0
msg_count_in_file = 0
total_msgs = 0
running = True

def _ensure_outdir():
    os.makedirs(OUT_DIR, exist_ok=True)

def _ts_filename():
    ts = int(time.time())
    return os.path.join(OUT_DIR, f"mqtt_batch_{ts}.ndjson")

def _open_new_file():
    global current_fp, current_path, opened_at, msg_count_in_file
    current_path = _ts_filename()
    current_fp = open(current_path, "a", encoding="utf-8",
buffering=1)
    opened_at = time.time()
    msg_count_in_file = 0
    print(f"[bridge] writing -> {current_path}")

def _close_file():
    global current_fp
    if current_fp is not None:
        try:
            current_fp.flush()
            current_fp.close()
        finally:
            print(f"[bridge] file closed: {current_path}")
```

```
            current_fp = None

def _should_rotate() -> bool:
    if current_fp is None:
        return True
    if ROTATE_EVERY_MSGS is not None and msg_count_in_file >=
ROTATE_EVERY_MSGS:
        return True
    if ROTATE_EVERY_SECONDS is not None and (time.time() -
opened_at) >= ROTATE_EVERY_SECONDS:
        return True
    if MAX_LINES_PER_FILE is not None and msg_count_in_file >=
MAX_LINES_PER_FILE:
        return True
    return False

def _graceful_exit(*_):
    global running
    running = False
    print("[bridge] exit signal received, closing...")

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print(f"[bridge] connected {MQTT_HOST}:{MQTT_PORT}
(rc=0)")
    else:
        print(f"[bridge] connect failed (rc={rc})")
    client.subscribe(MQTT_TOPIC, qos=1)
    print(f"[bridge] subscribed {MQTT_TOPIC}")

def on_message(client, userdata, msg):
    global current_fp, msg_count_in_file, total_msgs
    try:
        text = msg.payload.decode("utf-8", errors="strict")
        obj = json.loads(text)
        obj["_topic"] = msg.topic
        obj["_received_at"] =
datetime.now(timezone.utc).isoformat()

        if _should_rotate():
            _close_file()
            _open_new_file()

        current_fp.write(json.dumps(obj, ensure_ascii=False) +
"\n")
        msg_count_in_file += 1
        total_msgs += 1

        ts = obj.get("ts", "?")
        dev = obj.get("device_id", "?")
        print(f"[bridge] +1 ({msg_count_in_file} in file)
topic={msg.topic} ts={ts} device={dev}")
    except json.JSONDecodeError:
```

```
            print("[bridge] invalid JSON payload, ignored.")
        except Exception as e:
            print(f"[bridge] write error: {e}")

def main():
    _ensure_outdir()
    signal.signal(signal.SIGINT, _graceful_exit)
    try:
        signal.signal(signal.SIGTERM, _graceful_exit)
    except Exception:
        pass

    _open_new_file()

    client = mqtt.Client(client_id="", clean_session=True,
userdata=None, protocol=mqtt.MQTTv311)
    client.on_connect = on_connect
    client.on_message = on_message

    client.connect(MQTT_HOST, MQTT_PORT, keepalive=30)
    client.loop_start()

    try:
        while running:
            time.sleep(0.2)
    finally:
        client.loop_stop()
        client.disconnect()
        _close_file()
        print(f"[bridge] done. total messages: {total_msgs}")

if __name__ == "__main__":
    main()
```

ANEXO R. Job PySpark Structured Streaming – Ingesta de NDJSON y escritura en PostgreSQL

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit, when, to_timestamp,
get_json_object as j

IN_DIR = "/data/incoming_streaming"

PG_URL   = "jdbc:postgresql://localhost:5432/tfm"
PG_USER  = "usuario"
PG_PASS  = "contraseña"
PG_TABLE = "public.trc_observation"

spark = (
    SparkSession.builder
        .appName("tfm-clara-ingest")
        .getOrCreate()
)
spark.sparkContext.setLogLevel("WARN")

raw = (
    spark.readStream
         .format("text")
         .load(IN_DIR)
)

df = (
    raw
      .withColumn("ts_str",           j(col("value"), "$.ts"))
      .withColumn("device_id",        j(col("value"),
"$.device_id"))
      .withColumn("patient_id",       j(col("value"),
"$.patient_id"))
      .withColumn("evt_type_raw",     j(col("value"),
"$.evt_type"))
      .withColumn("crt_ms_str",       j(col("value"), "$.crt_ms"))
      .withColumn("rssi_str",         j(col("value"), "$.rssi"))
      .withColumn("_received_at_str", j(col("value"),
"$._received_at"))
)

df2 = (
    df
      .withColumn("evt_type",  when(col("evt_type_raw").isNull(),
lit("status")).otherwise(col("evt_type_raw")))
      .withColumn("ts",          to_timestamp(col("ts_str")))
      .withColumn("ingested_at",
to_timestamp(col("_received_at_str")))
      .withColumn("crt_ms",      col("crt_ms_str").cast("int"))
      .withColumn("rssi",        col("rssi_str").cast("int"))
```

```
        .withColumn("raw_payload", col("value"))

.select("ts","device_id","patient_id","evt_type","crt_ms","rssi","
raw_payload","ingested_at")
)

parquet_q = (
    df2.writeStream
        .format("parquet")
        .option("path", "/data/parquet/trc_observation")
        .option("checkpointLocation", "/data/_checkpoints/parquet")
        .outputMode("append")
        .start()
)

def to_pg(batch_df, batch_id):
    if batch_df.count() == 0:
        return
    (
        batch_df
          .write
          .format("jdbc")
          .option("url", PG_URL)
          .option("dbtable", PG_TABLE)
          .option("user", PG_USER)
          .option("password", PG_PASS)
          .option("driver", "org.postgresql.Driver")
          .option("stringtype","unspecified")
          .mode("append")
          .save()
    )

pg_q = (
    df2.writeStream
        .outputMode("append")
        .option("checkpointLocation",
"/data/_checkpoints/postgres")
        .foreachBatch(to_pg)
        .start()
)

spark.streams.awaitAnyTermination()
```

ANEXO S. Script SQL – Definición de la tabla de destino e índices del sistema de observaciones TRC

```sql
CREATE SCHEMA IF NOT EXISTS public;

DROP TABLE IF EXISTS public.trc_observation CASCADE;

CREATE TABLE public.trc_observation (
  id                  BIGSERIAL PRIMARY KEY,
  ts                  TIMESTAMPTZ NOT NULL,
  ingested_at         TIMESTAMPTZ NOT NULL DEFAULT now(),
  device_id           TEXT NOT NULL,
  patient_id          TEXT,
  evt_type            TEXT NOT NULL,
  crt_ms              INTEGER,
  recovery_thresh     DOUBLE PRECISION,
  color_baseline      DOUBLE PRECISION,
  color_min           DOUBLE PRECISION,
  color_recovery_pct  DOUBLE PRECISION,
  ambient_lux         DOUBLE PRECISION,
  force_peak_n        DOUBLE PRECISION,
  force_hold_ms       INTEGER,
  sqi                 INTEGER,
  rssi                INTEGER,
  qa                  JSONB,
  trace               JSONB,
  raw_payload         JSONB NOT NULL
);

CREATE INDEX IF NOT EXISTS trc_obs_ts_idx       ON
public.trc_observation (ts);
CREATE INDEX IF NOT EXISTS trc_obs_device_idx   ON
public.trc_observation (device_id);
CREATE INDEX IF NOT EXISTS trc_obs_evttype_idx  ON
public.trc_observation (evt_type);
```

ANEXO T. Consultas SQL de verificación de registros en DBeaver

```sql
SELECT COUNT(*)
FROM public.trc_observation;

SELECT
  ts, device_id, evt_type,
  (raw_payload::jsonb ->> 'crt_ms')::int AS crt_ms_from_raw,
  (raw_payload::jsonb ->> 'rssi')::int    AS rssi_from_raw,
  ingested_at
FROM public.trc_observation
ORDER BY ingested_at DESC
LIMIT 20;

SELECT
  ts, device_id, patient_id, ingested_at,
  raw_payload->>'evt_type'                    AS evt_type,
  (raw_payload->>'crt_ms')::int               AS crt_ms,
  (raw_payload->>'recovery_thresh')::numeric  AS recovery_thresh,
  (raw_payload->>'color_baseline')::numeric   AS color_baseline,
  (raw_payload->>'color_min')::numeric        AS color_min,
  (raw_payload->>'color_recovery_pct')::numeric AS
color_recovery_pct,
  (raw_payload->>'ambient_lux')::int          AS ambient_lux,
  (raw_payload->>'force_peak_n')::numeric     AS force_peak_n,
  (raw_payload->>'force_hold_ms')::int        AS force_hold_ms,
  (raw_payload->>'sqi')::int                  AS sqi,
  raw_payload->>'qa'                          AS qa_json_str,
  raw_payload->>'trace'                       AS trace_json_str
FROM public.trc_observation
WHERE raw_payload->>'evt_type' = 'crt'
ORDER BY ingested_at DESC
LIMIT 20;

SELECT
  ts,
  (raw_payload->>'qa')::jsonb ->> 'force_in_range'  AS
force_in_range,
  (raw_payload->>'qa')::jsonb ->> 'hold_in_range'   AS
hold_in_range
FROM public.trc_observation
WHERE raw_payload->>'evt_type' = 'crt'
ORDER BY ingested_at DESC
LIMIT 20;

SELECT
  ts,
  jsonb_array_length(((raw_payload->>'trace')::jsonb)->'ts_ms') AS
n_pts
FROM public.trc_observation
```

```
WHERE raw_payload->>'evt_type' = 'crt'
ORDER BY ingested_at DESC
LIMIT 20;

SELECT
  ts, device_id, ingested_at,
  (raw_payload->>'rssi')::int AS rssi
FROM public.trc_observation
WHERE COALESCE(raw_payload->>'evt_type','status') = 'status'
ORDER BY ingested_at DESC
LIMIT 20;
```

ANEXO U. Código Arduino – Firmware ESP32-C3 para visualización (publicación simulada cada 60 s)

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <time.h>
#include <esp_system.h>

const char* WIFI_SSID = "SSID";
const char* WIFI_PASS = "PASSWORD";

const char* MQTT_HOST = "test.mosquitto.org";
const uint16_t MQTT_PORT = 1883;

const char* DEVICE_ID  = "esp32_demo_01";
const char* PATIENT_ID = "anon_001";

String T_BASE   = String("tfm_clara_demo/") + DEVICE_ID + "/";
String T_STATUS = T_BASE + "status";
String T_CRT    = T_BASE + "crt";
String T_CMD    = T_BASE + "cmd";
String T_ACK    = T_BASE + "ack";
String T_LWT    = T_BASE + "lwt";

static const unsigned long STATUS_MS   = 10000;
static const unsigned long CRT_BASE_MS = 60000;

#define USE_JITTER 0
static const unsigned long JITTER_MAX_MS = 700;

#ifndef LED_BUILTIN
#define LED_BUILTIN 2
#endif

WiFiClient esp;
PubSubClient mqtt(esp);

unsigned long lastStatusMs = 0;
unsigned long lastCrtMs    = 0;
unsigned long nextCrtDueMs = 0;

String iso8601_utc_from(time_t t) {
  char buf[25];
  struct tm tm;
  gmtime_r(&t, &tm);
  strftime(buf, sizeof(buf), "%Y-%m-%dT%H:%M:%SZ", &tm);
  return String(buf);
}
```

```
bool waitForTime(uint16_t timeout_ms = 12000) {
  uint32_t t0 = millis();
  while ((time(nullptr) < 1700000000) && (millis() - t0 <
timeout_ms)) {
    delay(200);
  }
  return time(nullptr) >= 1700000000;
}

void mqttCallback(char* topic, byte* payload, unsigned int len) {
  String msg; msg.reserve(len);
  for (unsigned int i=0;i<len;i++) msg += (char)payload[i];
  String s = msg; s.trim(); s.toUpperCase();

  if (s == "LED_ON") {
    digitalWrite(LED_BUILTIN, HIGH);
    mqtt.publish(T_ACK.c_str(), "{\"ack\":\"LED_ON\"}");
  } else if (s == "LED_OFF") {
    digitalWrite(LED_BUILTIN, LOW);
    mqtt.publish(T_ACK.c_str(), "{\"ack\":\"LED_OFF\"}");
  } else if (s == "PING") {
    String ack = String("{\"ack\":\"PONG\",\"ms\":") +
String(millis()) + "}";
    mqtt.publish(T_ACK.c_str(), ack.c_str());
  } else {
    String ack = String("{\"ack\":\"UNKNOWN\",\"cmd\":\"") + msg +
"\"}";
    mqtt.publish(T_ACK.c_str(), ack.c_str());
  }
}

bool mqttConnect() {
  if (mqtt.connected()) return true;
  mqtt.setServer(MQTT_HOST, MQTT_PORT);
  mqtt.setCallback(mqttCallback);
  mqtt.setKeepAlive(5);
  mqtt.setBufferSize(4096);

  String clientId = "ESP32C3-" +
String((uint32_t)ESP.getEfuseMac(), HEX);
  bool ok = mqtt.connect(clientId.c_str(), nullptr, nullptr,
T_LWT.c_str(), 0, true, "{\"online\":false}");
  if (!ok) return false;

  mqtt.publish(T_LWT.c_str(), "{\"online\":true}", true);
  mqtt.subscribe(T_CMD.c_str());
  return true;
}

unsigned long computeNextCrtInterval() {
  if (!USE_JITTER) return CRT_BASE_MS;
```

```
  long jitter = (long)random(-(long)JITTER_MAX_MS,
(long)JITTER_MAX_MS + 1);
  long interval = (long)CRT_BASE_MS + jitter;
  if (interval < 10000) interval = 10000;
  return (unsigned long)interval;
}

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);

  Serial.begin(115200);
  delay(200);

  WiFi.mode(WIFI_STA);
  WiFi.setSleep(false);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  unsigned long t0 = millis();
  while (WiFi.status() != WL_CONNECTED && millis()-t0 < 20000) {
    delay(300);
  }
  if (WiFi.status() != WL_CONNECTED) return;

  configTzTime("UTC0", "pool.ntp.org", "time.nist.gov",
"time.google.com");
  waitForTime(15000);

  uint32_t seed = (uint32_t)esp_random() ^
(uint32_t)ESP.getEfuseMac() ^ (uint32_t)millis();
  randomSeed(seed);

  nextCrtDueMs = millis() + computeNextCrtInterval();
}

void loop() {
  if (WiFi.status() != WL_CONNECTED) { delay(200); return; }
  if (!mqttConnect()) { delay(500); return; }
  mqtt.loop();

  unsigned long nowMs = millis();
  time_t nowEpoch = time(nullptr);

  if (nowMs - lastStatusMs > STATUS_MS) {
    lastStatusMs = nowMs;
    DynamicJsonDocument doc(256);
    doc["v"] = 1;
    doc["evt_type"] = "status";
    doc["ts"] = iso8601_utc_from(nowEpoch);
    doc["device_id"] = DEVICE_ID;
    doc["patient_id"] = PATIENT_ID;
    doc["rssi"] = WiFi.RSSI();
    JsonObject data = doc.createNestedObject("data");
    data["ip"] = WiFi.localIP().toString();
```

```
    char out[256];
    size_t n = serializeJson(doc, out, sizeof(out));
    mqtt.publish(T_STATUS.c_str(), out, n);
  }

  if (nowMs >= nextCrtDueMs) {
    double recovery_thresh = 0.90;
    double color_baseline  = 0.78 + (random(-5,5) * 0.001);
    double color_min       = 0.15 + (random(-5,5) * 0.001);
    int    crt_ms          = 1700 + random(-300, 900);
    int    ambient_lux     = 90   + random(-60,80);
    double force_peak_n    = 6.5  + (random(-40,40) * 0.05);
    int    force_hold_ms   = 1000 + random(-120,120);
    int    sqi             = 80   + random(-20,15);

    String qa_str = String("{") +
      "\"force_in_range\":" + String((force_peak_n >= 5.0 &&
force_peak_n <= 8.0) ? "true" : "false") + "," +
      "\"hold_in_range\":"  + String((abs(force_hold_ms - 1000) <=
120) ? "true" : "false") + "," +
      "\"motion_high\":false," +
      "\"light_low\":"      + String((ambient_lux < 50) ? "true" :
"false") +
    "}";

    DynamicJsonDocument trace(384);
    JsonArray ts_ms  = trace.createNestedArray("ts_ms");
    JsonArray force  = trace.createNestedArray("force");
    JsonArray color  = trace.createNestedArray("color");
    int pts[8] = {0,200,400,600,800,1000,1500, crt_ms};
    for (int i=0;i<8;i++) {
      ts_ms.add(pts[i]);
      force.add(i<5 ? (0.6 + i*1.5) : (i==5 ? 7.0 : 0.4));
      double rec = min(1.0, (double)pts[i]/(double)crt_ms);
      double c = color_min + rec*(color_baseline - color_min);
      color.add(c);
    }

    DynamicJsonDocument doc(1500);
    doc["v"] = 1;
    doc["evt_type"] = "crt";
    doc["evt_id"] = String("crt-") +
String(random(100000,999999));
    doc["ts"] = iso8601_utc_from(nowEpoch);
    doc["device_id"] = DEVICE_ID;
    doc["patient_id"] = PATIENT_ID;
    doc["t_compress_start"] = iso8601_utc_from(nowEpoch - 3);
    doc["t_compress_end"]   = iso8601_utc_from(nowEpoch - 2);
    doc["t_recovery_end"]   = iso8601_utc_from(nowEpoch);
    doc["crt_ms"] = crt_ms;
    doc["recovery_thresh"] = recovery_thresh;
    doc["color_baseline"] = color_baseline;
    doc["color_min"] = color_min;
```

```
        doc["color_recovery_pct"] = 0.90;
        doc["ambient_lux"] = ambient_lux;
        doc["force_peak_n"] = force_peak_n;
        doc["force_hold_ms"] = force_hold_ms;
        doc["sqi"] = sqi;
        doc["qa"] = qa_str;
        doc["trace"] = trace.as<JsonObject>();
        char out[1600];
        size_t n = serializeJson(doc, out, sizeof(out));
        mqtt.publish(T_CRT.c_str(), out, n);

        nextCrtDueMs = nowMs + computeNextCrtInterval();
      }
    }
```

ANEXO V. PySpark – Reglas de alertado sobre eventos CRT para visualización

```
from pyspark.sql.functions import col, lit, array, struct,
to_json, explode

CRT_HIGH_MS = 2000
SQI_LOW_TH  = 75
LUX_LOW_TH  = 60
LUX_HIGH_TH = 900
F_MIN       = 0.7
F_MAX       = 2.8

rules = array(
    struct(
        lit("CRT_HIGH").alias("rule_id"),
        (col("crt_ms") > lit(CRT_HIGH_MS)).alias("trigger"),
        lit("critical").alias("severity"),
        to_json(struct(col("crt_ms"),
lit(CRT_HIGH_MS).alias("threshold_ms"))).alias("ctx")
    ),
    struct(
        lit("SQI_LOW").alias("rule_id"),
        (col("sqi") < lit(SQI_LOW_TH)).alias("trigger"),
        lit("warning").alias("severity"),
        to_json(struct(col("sqi"),
lit(SQI_LOW_TH).alias("threshold"))).alias("ctx")
    ),
    struct(
        lit("LIGHT_LOW").alias("rule_id"),
        (col("ambient_lux") < lit(LUX_LOW_TH)).alias("trigger"),
        lit("warning").alias("severity"),
        to_json(struct(col("ambient_lux"),
lit(LUX_LOW_TH).alias("threshold_low"))).alias("ctx")
    ),
    struct(
        lit("LIGHT_HIGH").alias("rule_id"),
        (col("ambient_lux") > lit(LUX_HIGH_TH)).alias("trigger"),
        lit("warning").alias("severity"),
        to_json(struct(col("ambient_lux"),
lit(LUX_HIGH_TH).alias("threshold_high"))).alias("ctx")
    ),
    struct(
        lit("FORCE_OUT").alias("rule_id"),
        ((col("force_peak_n") < lit(F_MIN)) | (col("force_peak_n")
> lit(F_MAX))).alias("trigger"),
        lit("warning").alias("severity"),
        to_json(struct(col("force_peak_n").alias("force_n"),
lit(F_MIN).alias("min_ok"),
lit(F_MAX).alias("max_ok"))).alias("ctx")
    ),
```

```
        struct(
            lit("ML_QUALITY").alias("rule_id"),

    col("ml_quality_label").isin("REPEAT","BORDERLINE").alias("trigger
    "),
            lit("warning").alias("severity"),
            to_json(struct(col("ml_quality_label"),
    col("ml_quality_score"),
    col("ml_model_version").alias("model"))).alias("ctx")
        )
    )

    alerts_exploded = (
        crt
          .withColumn("rules", rules)
          .withColumn("rule", explode(col("rules")))
          .filter(col("rule.trigger") == lit(True))
          .select(
              col("ts").alias("ts_event"),
              col("ingested_at"),
              col("device_id"),
              col("patient_id"),
              col("rule.rule_id").alias("rule_id"),
              col("rule.severity").alias("severity"),
              col("rule.ctx").alias("ctx")
          )
    )
```

ANEXO W. SQL – Tablas de agregados temporales y tabla de alertas

```
CREATE TABLE IF NOT EXISTS public.trc_observation_minute (
  id                BIGSERIAL PRIMARY KEY,
  window_start      TIMESTAMPTZ NOT NULL,
  window_end        TIMESTAMPTZ NOT NULL,
  device_id         TEXT NOT NULL,
  patient_id        TEXT,
  n_events          INTEGER NOT NULL,
  crt_ms_mean       DOUBLE PRECISION,
  crt_ms_p95        INTEGER,
  sqi_mean          DOUBLE PRECISION,
  created_at        TIMESTAMPTZ NOT NULL DEFAULT now()
);

CREATE INDEX IF NOT EXISTS trc_obs_minute_ws_idx  ON
public.trc_observation_minute (window_start);
CREATE INDEX IF NOT EXISTS trc_obs_minute_dev_idx ON
public.trc_observation_minute (device_id);

CREATE TABLE IF NOT EXISTS public.trc_alert (
  id              BIGSERIAL PRIMARY KEY,
  ts_event        TIMESTAMPTZ NOT NULL,
  ingested_at     TIMESTAMPTZ NOT NULL,
  device_id       TEXT NOT NULL,
  patient_id      TEXT,
  rule_id         TEXT NOT NULL,
  severity        TEXT NOT NULL,
  ctx             JSONB,
  created_at      TIMESTAMPTZ NOT NULL DEFAULT now()
);

CREATE INDEX IF NOT EXISTS trc_alert_ts_idx   ON public.trc_alert
(ts_event);
CREATE INDEX IF NOT EXISTS trc_alert_rule_idx ON public.trc_alert
(rule_id);
```

ANEXO X. Consultas SQL de verificación y métricas para visualización

```sql
SELECT
  ts, ingested_at, device_id,
  COALESCE(evt_type,'status') AS evt_type,
  crt_ms, sqi, ambient_lux, force_peak_n
FROM public.trc_observation
ORDER BY ingested_at DESC
LIMIT 20;

SELECT *
FROM public.trc_observation_minute
ORDER BY window_start DESC
LIMIT 20;

SELECT *
FROM public.trc_observation_5min
ORDER BY window_start DESC
LIMIT 20;

WITH m1 AS (
  SELECT
    date_trunc('minute', window_start) AS m,
    device_id,
    ROUND(crt_ms_mean::numeric,2) AS crt_mean_1m,
    crt_ms_p95 AS crt_p95_1m,
    n_events   AS n_1m
  FROM public.trc_observation_minute
  WHERE window_start >= NOW() - INTERVAL '30 minutes'
),
m5 AS (
  SELECT
    window_start AS m,
    device_id,
    ROUND(crt_ms_mean::numeric,2) AS crt_mean_5m,
    crt_ms_p95 AS crt_p95_5m,
    n_events   AS n_5m
  FROM public.trc_observation_5min
  WHERE window_start >= NOW() - INTERVAL '30 minutes'
)
SELECT
  COALESCE(m1.m, m5.m) AS bucket_ts,
  COALESCE(m1.device_id, m5.device_id) AS device_id,
  m1.crt_mean_1m, m5.crt_mean_5m,
  m1.crt_p95_1m,  m5.crt_p95_5m,
  m1.n_1m,        m5.n_5m
FROM m1
FULL OUTER JOIN m5
  ON m1.device_id = m5.device_id AND m1.m = m5.m
ORDER BY bucket_ts DESC, device_id;
```

```
SELECT
  ts_event, device_id, rule_id, severity, ctx::jsonb, created_at
FROM public.trc_alert
ORDER BY created_at DESC
LIMIT 20;

SELECT
  rule_id, COUNT(*) AS n
FROM public.trc_alert
GROUP BY 1
ORDER BY n DESC;
```

ANEXO Y. Consultas SQL para paneles de visualización en Grafana

```
-- ============================================================
-- PANEL 1: Estabilidad global del TRC
-- ============================================================
WITH ultimos AS (
  SELECT DISTINCT ON (o.patient_id)
    o.patient_id,
    CASE
      WHEN o.crt_ms IS NULL THEN NULL
      WHEN o.crt_ms >= 100 THEN o.crt_ms / 1000.0
      ELSE o.crt_ms::double precision
    END AS crt_s
  FROM public.trc_observation o
  WHERE o.evt_type = 'crt'
    AND o.patient_id IS NOT NULL AND o.patient_id <> ''
    AND o.crt_ms IS NOT NULL
    AND (o.raw_payload->>'ml_quality_label') = 'OK'
    AND $__timeFilter(COALESCE(o.ts, o.ingested_at))
  ORDER BY o.patient_id, COALESCE(o.ts, o.ingested_at) DESC
)
SELECT 'Normal (≤ 2 s)' AS estado,
       COUNT(*) FILTER (WHERE crt_s <= 2.0) AS pacientes
FROM ultimos
UNION ALL
SELECT 'Alto (> 2 s)' AS estado,
       COUNT(*) FILTER (WHERE crt_s > 2.0) AS pacientes;

-- ============================================================
-- PANEL 2: Estado actual del TRC por paciente
-- ============================================================
WITH ranked AS (
  SELECT
    COALESCE(o.ts, o.ingested_at) AS ts,
    o.device_id,
    o.patient_id,
    CASE
      WHEN o.crt_ms IS NULL THEN NULL
      WHEN o.crt_ms >= 100 THEN o.crt_ms::double precision
      ELSE o.crt_ms * 1000.0
    END AS trc_ms,
    o.sqi::double precision AS sqi,
    (o.qa->>'ml_quality_label') AS ml_quality_label,
    (o.qa->>'ml_quality_score')::double precision AS
ml_quality_score,
    LEAD(
      CASE
        WHEN o.crt_ms IS NULL THEN NULL
        WHEN o.crt_ms >= 100 THEN o.crt_ms::double precision
        ELSE o.crt_ms * 1000.0
```

```
      END
    ) OVER (PARTITION BY o.patient_id ORDER BY
COALESCE(o.ts,o.ingested_at) DESC) AS trc_prev_ms,
    ROW_NUMBER() OVER (PARTITION BY o.patient_id ORDER BY
COALESCE(o.ts,o.ingested_at) DESC) AS rn
  FROM public.trc_observation o
  WHERE o.evt_type = 'crt'
    AND $__timeFilter(COALESCE(o.ts, o.ingested_at))
    AND o.patient_id IS NOT NULL AND o.patient_id <> ''
),
ultima AS (
  SELECT patient_id, ts, device_id, trc_ms, trc_prev_ms, sqi,
ml_quality_label, ml_quality_score
  FROM ranked
  WHERE rn = 1
),
alerts AS (
  SELECT
    COALESCE(a.ts_event, a.ingested_at) AS t,
    a.patient_id,
    a.rule_id
  FROM public.trc_alert a
  WHERE a.rule_id IN
('SQI_LOW','SQL_LOW','LIGHT_LOW','LIGHT_HIGH','FORCE_OUT')
),
causas AS (
  SELECT
    u.patient_id,
    string_agg(
      DISTINCT CASE
        WHEN a.rule_id IN ('SQI_LOW','SQL_LOW') THEN 'Calidad de
señal baja'
        WHEN a.rule_id IN ('LIGHT_LOW','LIGHT_HIGH') THEN
'Iluminación fuera de rango'
        WHEN a.rule_id = 'FORCE_OUT' THEN 'Fuerza fuera de rango'
      END,
      ' . '
    ) AS causas_txt
  FROM ultima u
  LEFT JOIN alerts a
    ON a.patient_id = u.patient_id
   AND a.t BETWEEN (u.ts - INTERVAL '10 minutes') AND (u.ts +
INTERVAL '1 minutes')
  GROUP BY u.patient_id
),
calc AS (
  SELECT
    u.*,
    EXTRACT(EPOCH FROM (now() - u.ts)) AS delta_s,
    (u.trc_ms - COALESCE(u.trc_prev_ms, u.trc_ms)) / 1000.0 AS
var_s,
    CASE
      WHEN u.trc_prev_ms IS NULL OR u.trc_prev_ms = 0 THEN NULL
```

```
      ELSE (u.trc_ms - u.trc_prev_ms) / u.trc_prev_ms * 100.0
    END AS var_pct,
    c.causas_txt
  FROM ultima u
  LEFT JOIN causas c USING (patient_id)
)
SELECT
  c.patient_id AS "Paciente",
  CASE WHEN c.ml_quality_label = 'OK' THEN 'Correcta' ELSE 'No
fiable' END AS "Señal",
  CASE WHEN c.trc_ms >= 2000 THEN 'ALTO' ELSE 'OK' END AS "Estado
TRC",
  ROUND((c.trc_ms / 1000.0)::numeric, 2) AS "TRC (s)",
  CASE
    WHEN c.trc_prev_ms IS NULL THEN 'Sin histórico'
    WHEN ABS((c.trc_ms - c.trc_prev_ms)/1000.0) < 0.05 THEN '→
Estable'
    WHEN c.trc_ms < c.trc_prev_ms THEN '↑ Mejora'
    WHEN c.trc_ms > c.trc_prev_ms THEN '↓ Empeora'
    ELSE '→ Estable'
  END AS "Tendencia",
  to_char(c.ts, 'DD/MM/YY HH24:MI"h"') AS "Última medición",
  c.device_id AS "Dispositivo",
  NULLIF(c.causas_txt, '') AS "Detalle señal"
FROM calc c
WHERE c.patient_id IS NOT NULL AND c.patient_id <> ''
ORDER BY
  CASE WHEN c.trc_ms >= 2000 THEN 1 ELSE 2 END,
  c.delta_s ASC;

-- ============================================================
-- PANEL 3: Evolución temporal del TRC por paciente
-- ============================================================
SELECT
  $__time(COALESCE(o.ts, o.ingested_at)) AS "time",
  o.patient_id AS "metric",
  CASE
    WHEN o.crt_ms IS NULL THEN NULL
    WHEN o.crt_ms >= 100 THEN o.crt_ms / 1000.0
    ELSE o.crt_ms::double precision
  END AS "value"
FROM public.trc_observation o
WHERE o.evt_type = 'crt'
  AND o.patient_id IS NOT NULL AND o.patient_id <> ''
  AND (o.qa->>'ml_quality_label') = 'OK'
  AND $__timeFilter(COALESCE(o.ts, o.ingested_at))
ORDER BY 1, 2;

-- ============================================================
-- PANEL 4: Tendencia temporal del TRC
-- ============================================================
SELECT
  $__time(COALESCE(o.ts, o.ingested_at)) AS "time",
```

```
  o.patient_id AS "metric",
  CASE
    WHEN o.crt_ms IS NULL THEN NULL
    WHEN o.crt_ms >= 100 THEN o.crt_ms / 1000.0
    ELSE o.crt_ms::double precision
  END AS "value"
FROM public.trc_observation o
WHERE o.evt_type = 'crt'
  AND o.patient_id IS NOT NULL AND o.patient_id <> ''
  AND (o.qa->>'ml_quality_label') = 'OK'
  AND $__timeFilter(COALESCE(o.ts, o.ingested_at))
ORDER BY 1, 2;

-- ============================================================
-- PANEL 5: Alertas recientes
-- ============================================================
WITH ranked AS (
  SELECT
    COALESCE(a.ts_event, a.ingested_at) AS t,
    a.patient_id,
    a.device_id,
    a.rule_id,
    a.severity,
    a.ctx,
    ROW_NUMBER() OVER (PARTITION BY a.patient_id, a.rule_id ORDER
BY COALESCE(a.ts_event, a.ingested_at) DESC) AS rn
  FROM public.trc_alert a
  WHERE $__timeFilter(COALESCE(a.ts_event, a.ingested_at))
    AND a.rule_id IN
('CRT_HIGH','SQI_LOW','FORCE_OUT','LIGHT_LOW','LIGHT_HIGH')
),
ultima AS (
  SELECT * FROM ranked WHERE rn = 1
),
base AS (
  SELECT
    patient_id AS "Paciente",
    to_char(t AT TIME ZONE 'Europe/Madrid', 'DD/MM/YY
HH24:MI\"h\"') AS "Fecha y hora",
    CASE
      WHEN EXTRACT(EPOCH FROM (now() - t)) < 90 THEN 'ahora'
      WHEN EXTRACT(EPOCH FROM (now() - t)) < 3600 THEN
floor(EXTRACT(EPOCH FROM (now() - t))/60)::int || ' min'
      WHEN EXTRACT(EPOCH FROM (now() - t)) < 86400 THEN
floor(EXTRACT(EPOCH FROM (now() - t))/3600)::int || ' h'
      ELSE floor(EXTRACT(EPOCH FROM (now() - t))/86400)::int || '
d ' ||
           floor(mod(EXTRACT(EPOCH FROM (now() -
t))::bigint,86400)/3600)::int || ' h'
    END AS "Hace",
    CASE rule_id
      WHEN 'CRT_HIGH'   THEN 'TRC alto (≥ 2 s)'
      WHEN 'SQI_LOW'    THEN 'Calidad de señal baja'
```

```
      WHEN 'FORCE_OUT'  THEN 'Fuerza fuera de rango'
      WHEN 'LIGHT_LOW'  THEN 'Luz ambiente baja'
      WHEN 'LIGHT_HIGH' THEN 'Luz ambiente alta'
      ELSE 'Sistema / otros'
    END AS "Tipo de alerta",
    CASE
      WHEN severity IS NOT NULL THEN initcap(severity)
      WHEN rule_id = 'CRT_HIGH' THEN 'Crítico'
      ELSE 'Warning'
    END AS "Severidad",
    CASE
      WHEN ctx IS NOT NULL AND (ctx::jsonb ? 'crt_ms')
        THEN ROUND(((ctx::jsonb ->> 'crt_ms')::numeric / 1000.0),
2)
      ELSE NULL
    END AS "TRC (s)",
    CASE
      WHEN rule_id = 'CRT_HIGH' THEN 'Clínico – TRC elevado'
      WHEN rule_id = 'SQI_LOW' THEN 'Calidad de la señal – baja'
      WHEN rule_id = 'FORCE_OUT' THEN 'Fuerza fuera de rango'
      WHEN rule_id IN ('LIGHT_LOW','LIGHT_HIGH') THEN 'Iluminación
fuera de rango'
      ELSE 'Sistema / otros'
    END AS "Categoría de alerta",
    device_id AS "Dispositivo"
  FROM ultima
)
SELECT
  "Paciente",
  "Fecha y hora",
  "Hace",
  "Tipo de alerta",
  "Severidad",
  "TRC (s)",
  "Categoría de alerta",
  "Dispositivo"
FROM base
WHERE "Paciente" IS NOT NULL AND "Tipo de alerta" IS NOT NULL
ORDER BY
  CASE WHEN "Severidad" = 'Crítico' THEN 0 ELSE 1 END,
  "Fecha y hora" DESC;
```

ANEXO Z. Código Arduino – Firmware ESP32-C3 con TinyML para transmisión e inferencia local

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <time.h>
#include <esp_system.h>
#include "ml_inference.h"

const char* ML_CLASSES[] = {"BORDERLINE","OK","REPEAT"};

const char* WIFI_SSID = "SSID";
const char* WIFI_PASS = "PASSWORD";
const char* MQTT_HOST = "test.mosquitto.org";
const uint16_t MQTT_PORT = 1883;

const char* DEVICE_ID  = "esp32_demo_01";
const char* PATIENT_ID = "anon_001";

String T_BASE   = String("tfm_clara_demo/") + DEVICE_ID + "/";
String T_STATUS = T_BASE + "status";
String T_CRT    = T_BASE + "crt";
String T_CMD    = T_BASE + "cmd";
String T_ACK    = T_BASE + "ack";
String T_LWT    = T_BASE + "lwt";

static const unsigned long STATUS_MS   = 10000;
static const unsigned long CRT_BASE_MS = 60000;
#define USE_JITTER 0
static const unsigned long JITTER_MAX_MS = 700;

#ifndef LED_BUILTIN
#define LED_BUILTIN 2
#endif

WiFiClient esp;
PubSubClient mqtt(esp);

unsigned long lastStatusMs = 0;
unsigned long lastCrtMs    = 0;
unsigned long nextCrtDueMs = 0;

String iso8601_utc_from(time_t t) {
  char buf[25];
  struct tm tm;
  gmtime_r(&t, &tm);
  strftime(buf, sizeof(buf), "%Y-%m-%dT%H:%M:%SZ", &tm);
  return String(buf);
}
```

```
bool waitForTime(uint16_t timeout_ms = 12000) {
  uint32_t t0 = millis();
  while ((time(nullptr) < 1700000000) && (millis() - t0 <
timeout_ms)) delay(200);
  return time(nullptr) >= 1700000000;
}

bool mqttConnect() {
  if (mqtt.connected()) return true;
  mqtt.setServer(MQTT_HOST, MQTT_PORT);
  mqtt.setCallback([](char* topic, byte* payload, unsigned int
len) {
    String msg; msg.reserve(len);
    for (unsigned int i=0;i<len;i++) msg += (char)payload[i];
    String s = msg; s.trim(); s.toUpperCase();
    if (s == "LED_ON") { digitalWrite(LED_BUILTIN, HIGH);
mqtt.publish(T_ACK.c_str(), "{\"ack\":\"LED_ON\"}"); }
    else if (s == "LED_OFF") { digitalWrite(LED_BUILTIN, LOW);
mqtt.publish(T_ACK.c_str(), "{\"ack\":\"LED_OFF\"}"); }
    else if (s == "PING") { String ack =
String("{\"ack\":\"PONG\",\"ms\":") + String(millis()) + "}";
mqtt.publish(T_ACK.c_str(), ack.c_str()); }
  });
  mqtt.setKeepAlive(5);
  mqtt.setBufferSize(4096);
  String clientId = "ESP32C3-" +
String((uint32_t)ESP.getEfuseMac(), HEX);
  bool ok = mqtt.connect(clientId.c_str(), nullptr, nullptr,
T_LWT.c_str(), 0, true, "{\"online\":false}");
  if (!ok) return false;
  mqtt.publish(T_LWT.c_str(), "{\"online\":true}", true);
  mqtt.subscribe(T_CMD.c_str());
  return true;
}

unsigned long computeNextCrtInterval() {
  if (!USE_JITTER) return CRT_BASE_MS;
  long jitter = (long)random(-(long)JITTER_MAX_MS,
(long)JITTER_MAX_MS + 1);
  long interval = (long)CRT_BASE_MS + jitter;
  if (interval < 10000) interval = 10000;
  return (unsigned long)interval;
}

static float clamp01(float x){ return x<0?0:(x>1?1:x); }

static float curve_monotonicity_from_trace(const float* color,
const int* ts_ms, int n, int release_ms) {
  int inc=0, tot=0;
  for (int i=1;i<n;i++){ if (ts_ms[i] >= release_ms) { if
(color[i] >= color[i-1]) inc++; tot++; } }
  return (tot>0) ? float(inc)/float(tot) : 1.0f;
}
```

```
static float early_recovery_slope_from_trace(const float* color,
const int* ts_ms, int n, int release_ms) {
  const int t0 = release_ms, t1 = release_ms + 800;
  float c0 = -1, c1 = -1;
  for (int i=0;i<n;i++){ if (ts_ms[i] >= t0 && c0 < 0) c0 =
color[i]; if (ts_ms[i] >= t1) { c1 = color[i]; break; } }
  if (c0 < 0 || c1 < 0) return 0.0f;
  float slope = (c1 - c0) / 0.8f;
  return clamp01(slope);
}

static float residual_at_2s_from_trace(const float* color, const
int* ts_ms, int n, int release_ms, float g_baseline, float g_min)
{
  const int t2 = release_ms + 2000;
  float c2 = color[n-1];
  for (int i=0;i<n;i++){ if (ts_ms[i] >= t2) { c2 = color[i];
break; } }
  float rec = (c2 - g_min) / (g_baseline - g_min + 1e-6f);
  rec = clamp01(rec);
  return 1.0f - rec;
}

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
  Serial.begin(115200); delay(200);
  WiFi.mode(WIFI_STA);
  WiFi.setSleep(false);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  unsigned long t0 = millis();
  while (WiFi.status() != WL_CONNECTED && millis()-t0 < 20000)
delay(300);
  if (WiFi.status() != WL_CONNECTED) return;
  configTzTime("UTC0", "pool.ntp.org", "time.nist.gov",
"time.google.com");
  waitForTime(15000);
  uint32_t seed = (uint32_t)esp_random() ^
(uint32_t)ESP.getEfuseMac() ^ (uint32_t)millis();
  randomSeed(seed);
  nextCrtDueMs = millis() + computeNextCrtInterval();
}

void loop() {
  if (WiFi.status() != WL_CONNECTED) { delay(200); return; }
  if (!mqttConnect()) { delay(500); return; }
  mqtt.loop();

  unsigned long nowMs = millis();
  time_t nowEpoch = time(nullptr);

  if (nowMs - lastStatusMs > STATUS_MS) {
```

```
    lastStatusMs = nowMs;
    DynamicJsonDocument doc(256);
    doc["v"] = 1;
    doc["evt_type"] = "status";
    doc["ts"] = iso8601_utc_from(nowEpoch);
    doc["device_id"] = DEVICE_ID;
    doc["patient_id"] = PATIENT_ID;
    doc["rssi"] = WiFi.RSSI();
    JsonObject data = doc.createNestedObject("data");
    data["ip"] = WiFi.localIP().toString();
    char out[256]; size_t n = serializeJson(doc, out,
sizeof(out));
    mqtt.publish(T_STATUS.c_str(), out, n);
  }

  if (nowMs >= nextCrtDueMs) {
    double color_baseline  = 0.78 + (random(-5,5) * 0.001);
    double color_min       = 0.15 + (random(-5,5) * 0.001);
    int    crt_ms          = 1700 + random(-300, 900);
    int    ambient_lux     = 90   + random(-60,80);
    double force_peak_n    = 6.5  + (random(-40,40) * 0.05);
    int    force_hold_ms   = 1000 + random(-120,120);
    int    sqi             = 80   + random(-20,15);

    int ts_ms[8] = {0,200,400,600,800,1000,1500, crt_ms};
    float color[8];
    for (int i=0;i<8;i++){ double rec = min(1.0,
(double)ts_ms[i]/(double)crt_ms); color[i] = color_min +
rec*(color_baseline - color_min); }

    float force_mean_hold = 0.f; for (int i=0;i<5;i++)
force_mean_hold += (0.6f + i*1.5f); force_mean_hold /= 5.f;
    float curve_monotonicity =
curve_monotonicity_from_trace(color, ts_ms, 8, 1000);
    float early_recovery_slope =
early_recovery_slope_from_trace(color, ts_ms, 8, 1000);
    float residual_at_2s = residual_at_2s_from_trace(color, ts_ms,
8, 1000, color_baseline, color_min);

    float x_feat[11] = {
      force_mean_hold, (float)force_peak_n, (float)force_hold_ms,
(float)ambient_lux,
      (float)color_baseline, (float)color_min,
      curve_monotonicity, early_recovery_slope, residual_at_2s,
      (float)sqi, (float)crt_ms
    };

    MLResult ml = ml_predict(x_feat);
    const char* ml_label = ML_CLASSES[ml.class_index];
    float ml_score = ml.confidence;

    if (ml.class_index == 1) { digitalWrite(LED_BUILTIN, HIGH);
delay(60); digitalWrite(LED_BUILTIN, LOW); }
```

```
    else if (ml.class_index == 0) { digitalWrite(LED_BUILTIN,
HIGH); delay(80); digitalWrite(LED_BUILTIN, LOW); delay(80); }
    else { digitalWrite(LED_BUILTIN, HIGH); delay(200);
digitalWrite(LED_BUILTIN, LOW); delay(200); }

    DynamicJsonDocument doc(1024);
    doc["v"] = 1;
    doc["evt_type"] = "crt";
    doc["ts"] = iso8601_utc_from(nowEpoch);
    doc["device_id"] = DEVICE_ID;
    doc["patient_id"] = PATIENT_ID;
    doc["crt_ms"] = crt_ms;
    doc["ambient_lux"] = ambient_lux;
    doc["force_peak_n"] = force_peak_n;
    doc["force_hold_ms"] = force_hold_ms;
    doc["sqi"] = sqi;
    doc["ml_quality_label"] = ml_label;
    doc["ml_quality_score"] = ml_score;
    char out[1024]; size_t n = serializeJson(doc, out,
sizeof(out));
    mqtt.publish(T_CRT.c_str(), out, n);

    nextCrtDueMs = nowMs + computeNextCrtInterval();
  }
}
```