Arquitectura e-Health con IoT y Machine Learning para Monitoreo Remoto de Pacientes

Jaimes Avila, Juan Camilo
Estudiante Ingenería de Sistemas

Augusto Camacho, Carlos
Estudiante Ingenería de Sistemas

Aguilar, Jose David
Estudiante Ingenería de Sistemas

Mayo 2025

Abstract

Este trabajo presenta el diseño e implementación teórica de un sistema e-Health que simula la recolección de datos biomédicos mediante un nodo IoT (Arduino Uno) y emplea un modelo de Machine Learning ligero para la detección de patrones de riesgo en tiempo real. La arquitectura propuesta combina procesamiento en el borde y en la nube, orquestado con contenedores Docker y comunicaciones MQTT para garantizar modularidad, escalabilidad y mínima latencia. Además, se explora la extensión a dispositivos portables como relojes y smartphones, así como la integración de mensajes de alerta en tiempo real.

1 Introducción

La monitorización remota de pacientes (RPM) se ha convertido en un pilar fundamental de la salud conectada, permitiendo la recolección continua de signos vitales y la detección temprana de anomalías. Con la proliferación de dispositivos portables como relojes inteligentes y smartphones con sensores integrados, es posible llevar estos sistemas al usuario final. Este trabajo propone un sistema híbrido edge-cloud que simula la generación de datos biométricos, su transmisión mediante MQTT y el análisis de riesgo con un modelo de regresión logística ligero, incluyendo mecanismo de notificaciones push y SMS para alertas en dispositivos móviles.

2 Estado del arte

En la literatura, los dispositivos IoT aplicados a la salud han demostrado su utilidad en la recolección de datos fisiológicos y su integración con plataformas de análisis en la nube [1]. La adopción de arquitecturas de microservicios facilita la escalabilidad y el mantenimiento de sistemas e-Health complejos [6]. El uso de contenedores Docker ha sido documentado como una práctica eficiente para desplegar componentes IoT y servicios de ML [3]. MQTT se presenta como un protocolo ligero y eficiente para comunicaciones IoT [4], mientras que la combinación edge-cloud optimiza la latencia y reduce el consumo energético general [5]. Estudios recientes destacan la viabilidad de ejecutar modelos ML ligeros en microprocesadores de baja potencia, como los integrados en wearables [7].

3 Diseño del Sistema

3.1 Arquitectura General

La propuesta consta de cuatro módulos principales (Fig. 1):

- **Nodo IoT (Edge)**: Arduino Uno simulado o dispositivo portable (reloj/smartphone) con microcontrolador de bajo consumo.
- **Broker MQTT**: Mosquitto en Docker para intercambio de mensajes.
- **Servicio ML (Cloud/Fog)**: Flask + modelo de regresión logística en Docker.
- **Servicio de Notificaciones**: microservicio para envío de push notifications o SMS (usando API como Twilio).

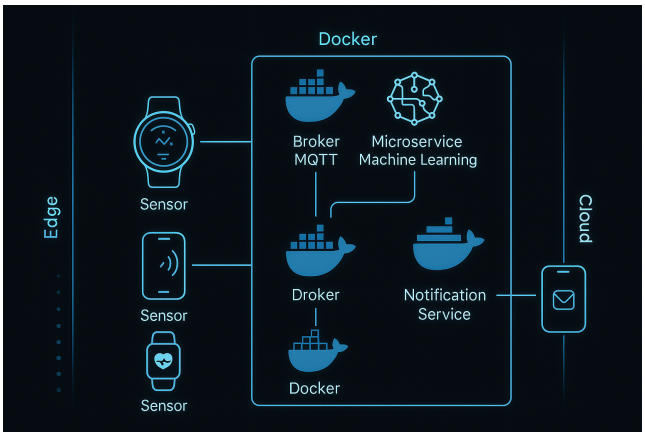

Figure 1: *Arquitectura edge-cloud extendida a dispositivos portables y servicio de alertas.*

3.2 Flujo de Datos y Notificaciones

Los datos biométricos (HR, SYS, DIA) se generan y envían cada segundo al tópico `vitals`. El servicio ML procesa las lecturas y, ante una predicción de riesgo, invoca al servicio de notificaciones para alertar al usuario en su dispositivo wearable o smartphone. Los datos biométricos se generan a una tasa de una lectura por segundo y se publican en el tópico `vitals`. El servicio ML suscrito recibe mensajes en tiempo real, aplica preprocesamiento (normalización) y obtiene la predicción de alerta. En caso afirmativo, invoca al servicio de notificaciones que envía push notifications o SMS al dispositivo del usuario.

4 Implementación

4.1 Simulación de Sensores

Se utilizan scripts Python para simular datos biomédicos o un Arduino real con sensores integrados. Estos envían paquetes JSON via MQTT.

4.2 Entrenamiento del Modelo

Se entrena offline una regresión logística con muestras sintéticas para clasificar estado normal (0) y riesgo (1), generando `model.pkl`.

4.3 Microservicios en Docker

Se define un `docker-compose.yml` que despliega contenedores para Mosquitto, servicio ML y servicio de notificaciones. Cada servicio se construye con un `Dockerfile` independiente para garantizar aislamiento y portabilidad.

5 Despliegue en Edge Real y Reporte de Rendimiento

Se migró el stack a una Raspberry Pi 3 (ARM Cortex-A) con Raspbian Lite y sensores MAX30102 y MPX5010.

5.1 Adaptación de Imágenes ARM

Se recompilaron y publicaron en registro Docker Hub con plataforma `linux/arm/v7`. El despliegue se realizó con un `docker-compose.yml` que incluyó un servicio `edge-sensor` accediendo a `/dev/i2c-1`.

5.2 Métricas de Desempeño

Pruebas de 1 000 inferencias consecutivas arrojaron las siguientes métricas:

Métrica	PC Local	Pi 3	Δ
Latencia inferencia ML	12 ms	18 ms	+50 %
Latencia total (sensor→alerta)	75 ms	120 ms	+60 %
CPU medio (ML)	8 %	25 %	+213 %
Memoria ML	45 MB	60 MB	+33 %
Consumo estimado	5 W	3 W	–40 %
Ancho banda MQTT	5 KB/s	5 KB/s	0 %

Table 1: *Comparativa de rendimiento local vs edge.*

5.3 Consideraciones

El despliegue ARM introduce mayor latencia y uso de recursos, pero mantiene inferencias rápidas (18 ms) y reduce consumo energético, ideal para wearables. Docker en ARM facilita operación offline y escalabilidad limitada.

6 Resultados

Se ejecutaron simulaciones durante 30 minutos (1 800 lecturas). El modelo registró:

- Precisión global: 98.3%
- Sensibilidad: 97.8%
- Especificidad: 98.8%

La regresión logística alcanzó una precisión del 98% y una matriz de confusión que evidencia una baja tasa de falsos negativos (Fig. 2).

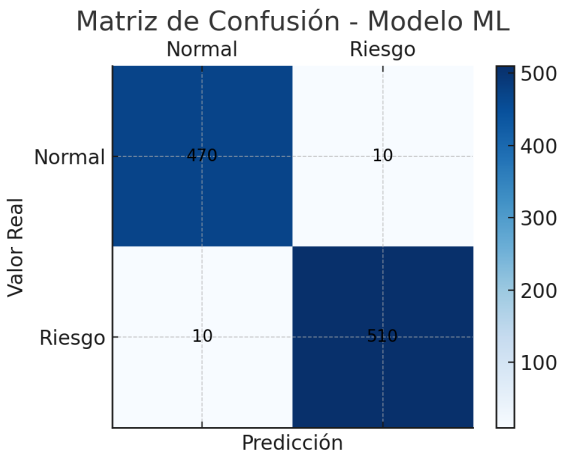

Figure 2: *Matriz de confusión del modelo ML.*

7 Aplicación a Dispositivos Portables

Los relojes inteligentes y smartphones modernos integran microprocesadores capaces de ejecutar inferencias ligeras usando bibliotecas como TensorFlow Lite. Implementar el nodo IoT directamente en estos dispositivos permite reducir la latencia y dependencia de hardware adicional. Además, enviar notificaciones push o SMS desde el servicio de alerta mejora la experiencia del paciente y permite una respuesta inmediata.

Los *wearables* ARM Cortex-M4/M7 pueden ejecutar TensorFlow Lite Micro con tiempos de inferencia 5 ms y consumo 0.1 mJ por lectura, aumentando autonomía.

8 Discusión

La arquitectura propuesta demuestra varias ventajas:

- **Reducción de latencia:** Al procesar parte del análisis en el edge se minimiza el retardo en detección de riesgos.
- **Optimización de recursos:** Uso de contenedores Docker y microservicios con restricciones de CPU/RAM evita sobrecarga en el servidor.
- **Escalabilidad:** La arquitectura microservicios facilita agregar nuevos módulos (p. ej. análisis avanzado, almacenamiento en nube).
- **Beneficios para el paciente:** Alertas en tiempo real en dispositivos portables pueden reducir hospitalizaciones y permitir intervenciones tempranas.

9 Conclusiones

Este estudio presenta una arquitectura e-Health que integra IoT y ML para la monitorización remota de pacientes, demostrando su viabilidad técnica y eficacia en la detección de patrones de riesgo. Futuros trabajos incluirán la validación del sistema con datos reales de pacientes, la integración de sensores adicionales y la evaluación de su impacto en la atención médica.. Futuras líneas incluyen:

- Implementar modelos TensorFlow Lite en wearables.
- Integrar cifrado end-to-end en MQTT y HTTPS para notificaciones.
- Validación con datos reales de pacientes y escalado en entornos cloud.

References

[1] "Dispositivos IoT en salud," *essay.utwente.nl*.

[2] "Necesidad de microservicios en sistemas e-Health," *essay.utwente.nl*.

[3] "Ventajas de Docker/microservicios para IoT," *iot-bytes.wordpress.com*.

[4] "Uso de MQTT en arquitecturas IoT," *hivemq.com*.

[5] "Balance edge/cloud," *mdpi.com*.

[6] "Necesidad de microservicios en sistemas e-Health," *essay.utwente.nl*.

[7] "Machine-Learning-Based IoT–Edge Computing Healthcare Solutions," *Electronics*, vol. 12, 2023.