EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Doctorado

PLATAFORMA PARA LA CONFIGURACIÓN AUTOMÁTICA DE DISPOSITIVOS BASADOS EN ARDUINO AL INTERNET DE LAS COSAS

presentada por

MC. Juan José Flores Sedano

como requisito para la obtención del grado de

Doctor en Ciencias de la Computación

Director de tesis

Dr. Hugo Estrada Esquivel

Codirectora de tesis

Dra. Alicia Martínez Rebollar

Cuernavaca, Morelos, México. Agosto del 2026

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

cenidet	ACEPTACIÓN DE IMPRESIÓN DEL DOCUMENTO DE TESIS DOCTORAL	Código: CENIDET-AC-006-D20
		Revisión: O
	Referencia a la Norma ISO 9001:2008 7.1, 7.2.1, 7.5.1, 7.6, 8.1, 8.2.4	Página 1 de 1

Cuernavaca, Mor., a 23 de junio de 2026

DR. CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO
P R E S E N T E

AT'n: DRA. ANDREA MAGADÁN SALAZAR
PRESIDENTA DEL CLAUSTRO DOCTORAL DEL
DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

Los abajo firmantes, miembros del Comité Tutorial del estudiante **M.C. JUAN JOSÉ FLORES SEDANO** manifiestan que después de haber revisado el documento de tesis titulado **"Plataforma para la configuración automática de dispositivos basados en arduino al internet de las cosas"**, realizado bajo la dirección del **Dr. Hugo Estrada Esquivel** y codirigido por la **Dra. Alicia Martínez Rebollar,** el trabajo se ACEPTA para proceder a su impresión.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Conocimiento y Tecnología al Servicio de México"

DR. HUGO ESTRADA ESQUIVEL
TECNM/CENIDET

DRA. ALICIA MARTÍNEZ REBOLLAR
TECNM/CENIDET

DR. JAVIER ORTIZ HERNÁNDEZ
TECNM/CENIDET

DRA. MARÍA YASMÍN HERNÁNDEZ PÉREZ
TECNM/CENIDET

DR. CARLOS DANIEL GARCÍA BELTRÁN
TECNM/CENIDET

DR. MIGUEL GONZÁLEZ MENDOZA
TECNOLÓGICO DE MONTERREY, CAMPUS ESTADO DE MÉXICO

c.c.p. C Verónica Sotelo Boyás/ Jefa del Departamento de Servicios Escolares
c.c.p. C Nóe Alejandro Castro Sánchez / Jefe del Departamento de Ciencias Computacionales
c.c.p. Expediente

Educación
Secretaría de Educación Pública

Centro Nacional de Investigación y Desarrollo Tecnológico
Subdirección Académica

Cuernavaca Mor., 26/junio/2026
Oficio No. SAC/189/2026
Asunto: Autorización de impresión de tesis

JUAN JOSÉ FLORES SEDANO
CANDIDATO AL GRADO DE DOCTOR EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"Plataforma para la configuración automática de dispositivos basados en arduino al internet de las cosas"**, ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un cordial saludo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica.
"Conocimiento y Tecnología al servicio de México"

EDUCACIÓN
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SUBDIRECCIÓN ACADÉMICA

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO

c.c.p. Departamento de Ciencias Computacionales
Departamento de Servicios Escolares

CMAZ/mrz

2026 año de Margarita Maza

cenidet

Interior Internado Palmira S/N, Col. Palmira,
C.P. 62490, Cuernavaca, Morelos Tel. 01 (777) 3627770, ext. 4106
e-mail: acad_cenidet@tecnm.mx | cenidet.tecnm.mx

Dedicatoria

Este trabajo lo dedico a mi familia, amigos y profesores que durante toda mi carrera me han apoyado de manera incondicional, jamás dejaron que me diera por vencido y sin ellos este, que hasta ahora es el mayor logro de mi vida no habría sido posible. En especial va dedicado a la mi madre que durante toda mi vida me apoyó y sin ella no estaría aquí, a mis hermanas que en momentos de estrés y desesperación me brindaron todo su apoyo, a mi hermano que siempre está cuando más lo necesito, también quiero dedicar este logro a mi segunda madre mi abuela quien ha estado en los momentos más difíciles de mi vida y a pesar de todo sigue conmigo, a mi tio Angel quien desde la universidad me apoyó desvelándose conmigo mientras hacía mi tarea, por último quiero hacer una dedicatoria a mi padre que desde el cielo está mirando mi más grande logro, tengo la esperanza de que el este feliz y orgulloso de mi.

Agradecimientos

Quiero agradecer a Dios por darme la oportunidad de vivir esta maravillosa experiencia, agradecer a los profesores que siempre estuvieron para darme consejos y ánimos durante toda la carrera, en especial al Dr. Hugo Estrada Esquivel y a la Dra. Alicia Martínez Rebollar quien durante los últimos 2 años de mi carrera me han apoyado y brindado la oportunidad de trabajar con ellos y experimentar el mundo de la investigación, si no fuera por ellos no me hubiera animado a estudiar un posgrado. También quiero agradecer a mis amigos Manuel Erazo, Daza, y Juan Antonio, quienes hicieron de la carrera la más divertida e interesante, con quienes pasé los mejores momentos de mi vida, agradezco al SECIHTI por la beca otorgada y al TecNM por facilitar sus instalaciones. Por último, pero no menos importante quiero agradecer a mi familia, a Mi madre quien a pesar de todos los dolores de cabeza me ha apoyado de manera incondicional, a mis hermanas Yadira y Rebeca quienes siempre están cuando las necesito y a mi hermano Marcos quien me ha enseñado a siempre seguir adelante y a los demás miembros de mi familia quienes en todo momento estuvieron conmigo apoyándome y dándome ánimos, mil, mil gracias a todos ustedes por estar conmigo.

Resumen

Esta tesis propone y formaliza la arquitectura de un Agente Inteligente de IoT embebido, definido mediante la función de decisión f: P × H → A, capaz de percibir su entorno mediante sensores, ejecutar inferencias locales con modelos de inteligencia artificial y comunicar sus resultados de forma estandarizada. Sobre esta base, se presenta una plataforma para la generación automática de dichos agentes en dispositivos Arduino. El trabajo aborda la brecha existente entre la inferencia de inteligencia artificial ligera (TinyML) y la integración estandarizada con plataformas IoT, las cuales han avanzado de forma paralela sin converger en una solución unificada y accesible.

Se propone y formaliza un agente inteligente embebido mediante la función de decisión f: P × H → A, donde P representa el conjunto de percepciones capturadas por sensores, H el historial interno representado mediante el estándar NGSI v2, y A el conjunto de acciones posibles condicionadas por las capacidades del hardware. La plataforma desarrollada en R/Shiny automatiza el proceso completo: desde el entrenamiento de modelos de aprendizaje automático (Random Forest, Red Neuronal cuantizada y Árbol de Decisión) hasta la generación automática del firmware desplegable en dispositivos Arduino, sin requerir programación manual por parte del usuario.

La validación experimental se realizó en dos casos de estudio: detección de caídas en adultos mayores, logrando una exactitud del 92 % con latencias de 25–35 ms en un Arduino Nano 33 BLE; y estimación de evapotranspiración de referencia (ETo) en agricultura, alcanzando R^2=0.925 con RMSE=0.195. Las pruebas de confiabilidad operativa de 24 horas demostraron R ≥ 0.9994 en los cuatro dispositivos evaluados.

Los resultados validan la viabilidad de ejecutar inferencia de modelos de IA directamente en microcontroladores de bajo costo, integrarse con plataformas IoT estandarizadas como FIWARE mediante NGSI v2, y automatizar la generación del agente sin intervención técnica especializada. La investigación alcanza el nivel TRL 5 con avance parcial hacia TRL 6.

Palabras clave: Agentes inteligentes embebidos, TinyML, Internet de las Cosas, FIWARE, NGSI v2, Arduino, Aprendizaje automático ligero, Detección de caídas, Evapotranspiración.

Abstract

This thesis proposes and formalizes the architecture of an embedded IoT Intelligent Agent, defined through the decision function f: P × H → A, capable of perceiving its environment through sensors, executing local inferences with artificial intelligence models, and communicating its results in a standardized manner. Building on this foundation, a platform for the automatic generation of such agents on Arduino devices is presented. The work addresses the existing gap between lightweight artificial intelligence inference (TinyML) and standardized integration with IoT platforms, which have advanced in parallel without converging into a unified and accessible solution.

An embedded intelligent agent is proposed and formalized through the decision function f: P × H → A, where P represents the set of perceptions captured by sensors, H the internal history represented using the NGSI v2 standard, and A the set of possible actions conditioned by hardware capabilities. The platform developed in R/Shiny automates the complete process from training machine learning models to the automatic generation of firmware deployable on Arduino devices, without requiring manual programming by the user.

Experimental validation was conducted in two case studies: fall detection in elderly adults, achieving 92% accuracy with latencies of 25–35 ms on an Arduino Nano 33 BLE; and reference evapotranspiration (ETo) estimation in agriculture, reaching R^2=0.925 with RMSE=0.195. Twenty-four-hour operational reliability tests demonstrated R ≥ 0.9994 across all four evaluated devices.

The results validate that it is possible to execute machine learning model inference directly on low-cost microcontrollers, integrate natively with standardized IoT platforms such as FIWARE via NGSI v2, and automate the generation of the complete agent without specialized technical intervention. The research achieves TRL 5 with partial progress toward TRL 6.

Keywords: Embedded intelligent agents, TinyML, Internet of Things, FIWARE, NGSI v2, Arduino, Lightweight machine learning, Fall detection, Evapotranspiration.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1 – Introducción

1.1 Introducción

El avance del Internet de las Cosas (IoT, por sus siglas en inglés) ha transformado la forma en que los dispositivos físicos interactúan con su entorno y con plataformas digitales. A través de sensores, actuadores, microcontroladores y redes de comunicación, el IoT ha permitido el desarrollo de sistemas capaces de capturar información del mundo real, transmitirla a la nube y apoyar la automatización de procesos en múltiples dominios, como salud, agricultura, monitoreo ambiental, automatización industrial y ciudades inteligentes.

Dentro de este ecosistema, los dispositivos basados en microcontroladores de bajo costo, como Arduino, han cobrado gran relevancia debido a su accesibilidad, facilidad de uso y amplia adopción en prototipos, sistemas embebidos y soluciones IoT. Sin embargo, a pesar de su popularidad, estos dispositivos suelen ser utilizados únicamente como nodos de adquisición y transmisión de datos, limitando su participación a funciones básicas de sensado y comunicación.

En paralelo, el crecimiento reciente de enfoques como TinyML y Edge AI ha abierto la posibilidad de ejecutar modelos de inteligencia artificial directamente en hardware con capacidades limitadas de procesamiento, permitiendo que los dispositivos no solo recopilen datos, sino que también los interpreten localmente y generen respuestas útiles en tiempo real. Esta capacidad resulta particularmente relevante en escenarios donde la latencia, la privacidad, el consumo energético o la conectividad intermitente hacen inviable depender completamente de la nube.

No obstante, integrar capacidades de inteligencia artificial embebida dentro de un flujo funcional completo de IoT sigue siendo una tarea compleja. En la práctica, desarrollar una solución de este tipo implica resolver múltiples aspectos de manera simultánea: adquisición de datos, entrenamiento y adaptación de modelos, despliegue en hardware con capacidades limitadas de procesamiento, estructuración de la información y comunicación interoperable con plataformas IoT.

En este contexto, la presente investigación propone y formaliza el concepto de Agente Inteligente de IoT como un componente software embebido capaz de adquirir datos del entorno mediante sensores, ejecutar modelos de inteligencia artificial de manera local y comunicar sus resultados de forma interoperable con plataformas IoT. Asimismo, se propone el diseño e implementación de una plataforma para la generación automática de dichos agentes en dispositivos Arduino, particularmente bajo un enfoque orientado a FIWARE y modelos de datos estandarizados.

La investigación parte de la premisa de que el valor de un dispositivo IoT no debe limitarse únicamente a su conectividad, sino a su capacidad de interpretar información del entorno, generar inferencias útiles y actuar o comunicar resultados de forma contextualizada. Desde esta perspectiva, el trabajo se sitúa en la intersección entre IoT, inteligencia artificial embebida, agentes inteligentes y sistemas distribuidos, proponiendo una solución que busca reducir la complejidad técnica del desarrollo y ampliar la viabilidad de soluciones inteligentes en dispositivos de bajos recursos.

1.2 Contexto del problema

El Internet de las Cosas ha favorecido la aparición de una gran diversidad de dispositivos capaces de conectarse a redes y plataformas digitales para intercambiar información en tiempo real. Actualmente, sensores, actuadores, dispositivos portátiles y sistemas embebidos forman parte de entornos conectados en los que la adquisición y transmisión de datos se ha convertido en una práctica común.

En este panorama, los microcontroladores como Arduino han adquirido una gran relevancia por su bajo costo, facilidad de programación y amplia disponibilidad. Estas características los convierten en una alternativa atractiva para el desarrollo de soluciones IoT en contextos académicos, industriales y sociales. Sin embargo, el uso de estos dispositivos suele limitarse a tareas básicas como la lectura de sensores, el control de actuadores o la transmisión de información hacia plataformas externas.

De manera paralela, el término *dispositivo inteligente* se ha popularizado ampliamente en el mercado tecnológico. Actualmente, productos como focos, apagadores, enchufes, cerraduras o cámaras son frecuentemente comercializados como *smart devices*. No obstante, en muchos casos, la inteligencia atribuida a estos dispositivos se reduce a la capacidad de ser controlados de forma remota mediante una aplicación móvil o a la ejecución de reglas predefinidas programadas por el usuario.

Esta situación genera una confusión conceptual importante, ya que no todo dispositivo conectado o automatizado puede considerarse realmente inteligente desde una perspectiva computacional. En muchos casos, estos dispositivos no interpretan su entorno, no analizan patrones de comportamiento y no generan inferencias basadas en modelos de conocimiento adquiridos a partir de datos.

Por otra parte, los avances recientes en aprendizaje automático embebido han demostrado que es posible ejecutar modelos ligeros de inteligencia artificial directamente en microcontroladores de recursos restringidos. Sin embargo, la mayoría de estas implementaciones suelen centrarse únicamente en la ejecución aislada del modelo, sin resolver de manera integral su incorporación dentro de una arquitectura IoT funcional, interoperable y reutilizable.

En consecuencia, surge la necesidad de transitar desde una visión centrada en dispositivos simplemente conectados hacia una visión basada en agentes inteligentes embebidos, capaces

no solo de capturar y transmitir datos, sino también de interpretarlos localmente y participar de manera activa en la toma de decisiones dentro de un ecosistema IoT.

1.3 De los dispositivos conectados a los agentes inteligentes

Gran parte de estos sistemas operan como dispositivos conectados o automatizados, cuya funcionalidad principal consiste en recibir comandos remotos, ejecutar acciones previamente definidas o transmitir información a una plataforma en la nube. Por ejemplo, una lámpara inteligente puede encenderse o apagarse desde un teléfono móvil, pero esa acción generalmente depende de una orden explícita del usuario o de una regla previamente programada. En este caso, el dispositivo no interpreta activamente su contexto ni toma decisiones derivadas del análisis de datos del entorno.

Esta distinción es relevante para el presente trabajo, ya que la propuesta desarrollada en esta tesis no se limita a la conectividad o automatización remota del dispositivo. En esta investigación, un sistema basado en Arduino incorpora un agente inteligente embebido, entendido como un componente software capaz de adquirir información del entorno, procesarla mediante un modelo de inteligencia artificial previamente entrenado y producir una salida útil para apoyar la toma de decisiones locales o remotas.

La inteligencia del sistema propuesto no debe interpretarse como inteligencia general o aprendizaje autónomo ilimitado, sino como una inteligencia funcional embebida, orientada a tareas específicas. Esto significa que el dispositivo puede:

- Percibir el entorno, mediante sensores físicos.
- Procesar e interpretar datos localmente, utilizando un modelo de aprendizaje automático entrenado previamente.
- Generar una inferencia útil, como una clasificación, predicción o detección de eventos.
- Ejecutar una acción o comunicar un resultado contextualizado, de acuerdo con la inferencia obtenida.

Bajo esta perspectiva, la diferencia fundamental entre un dispositivo conectado y un agente inteligente radica en que el segundo no depende exclusivamente de comandos externos o de reglas rígidas, sino que incorpora una capacidad de inferencia local que le permite responder al entorno de forma contextual.

En consecuencia, dentro de esta tesis, un dispositivo puede considerarse inteligente no solo por su capacidad de conectarse a Internet, sino por su capacidad de analizar información del

entorno, generar inferencias útiles y actuar o comunicar resultados en función de dichas inferencias. Esta distinción es fundamental para sustentar el concepto de agente inteligente embebido propuesto en esta investigación.

1.4 Planteamiento del problema

A pesar del crecimiento del Internet de las Cosas y del uso extendido de microcontroladores como Arduino en aplicaciones de monitoreo y automatización, la integración efectiva de estos dispositivos dentro de ecosistemas IoT sigue siendo una tarea compleja, fragmentada y altamente dependiente de configuración manual.

En la práctica, conectar un dispositivo basado en Arduino a una plataforma IoT implica resolver múltiples aspectos técnicos, entre ellos la selección del protocolo de comunicación, la definición del formato de intercambio de datos, la estructuración de la información bajo un modelo interoperable, la programación del envío hacia una plataforma externa y, en algunos casos, la incorporación de mecanismos de seguridad o autenticación.

Esta complejidad incrementa el esfuerzo de desarrollo, limita la reutilización de las soluciones y dificulta su adopción en escenarios donde se requiere rapidez de implementación, bajo costo y flexibilidad.

Además, esta problemática no se reduce únicamente a la conectividad. En muchos contextos reales, un dispositivo IoT no solo debe transmitir información, sino también interpretar su entorno y generar respuestas útiles a partir de los datos capturados. Por ejemplo, en un sistema de detección de caídas, el dispositivo debe ser capaz de identificar un patrón anómalo en el movimiento; en un sistema agrícola, debe poder estimar una variable relevante del entorno a partir de sensores físicos. En ambos casos, la utilidad del sistema depende de que el dispositivo no sea solo un emisor de datos, sino un nodo capaz de producir inferencia local.

Aunque los avances recientes en TinyML y Edge AI han permitido ejecutar modelos de inteligencia artificial en microcontroladores de recursos restringidos, la mayoría de las soluciones existentes se centran en el despliegue aislado del modelo y no en la generación automática de un agente inteligente completo. Es decir, no suelen integrar de forma conjunta la adquisición de datos, la inferencia embebida, la estructuración contextual de la información y la comunicación interoperable con plataformas IoT como parte de un mismo flujo funcional.

Como consecuencia, existe una brecha entre el desarrollo de modelos de IA embebida y la construcción de soluciones IoT funcionales, reutilizables e interoperables para dispositivos de bajo costo. Esta brecha limita el potencial de los microcontroladores como nodos inteligentes distribuidos y reduce su papel a funciones de sensado o transmisión, desaprovechando su posible participación en tareas de análisis local y toma de decisiones contextual.

Ante esta situación, surge la necesidad de crear y formalizar el concepto de Agente Inteligente de IoT como un agente embebido capaz de percibir su entorno mediante sensores, ejecutar inferencias locales mediante modelos de inteligencia artificial y comunicar sus resultados de forma estandarizada e interoperable. Asimismo, surge la necesidad de desarrollar una plataforma que permita automatizar la generación de esos agentes inteligentes embebidos para dispositivos Arduino, integrando en un mismo proceso la adquisición de datos, el entrenamiento

y adaptación de modelos de IA, la generación del componente embebido y su comunicación interoperable con plataformas IoT.

1.5 Pregunta de investigación

La pregunta central que guía esta investigación es la siguiente:

> ¿Cómo diseñar e implementar una plataforma capaz de generar automáticamente agentes inteligentes embebidos en dispositivos Arduino, que integren modelos de inteligencia artificial, operen bajo restricciones de hardware y se comuniquen de manera interoperable con plataformas del Internet de las Cosas?

De manera complementaria, esta investigación también se apoya en las siguientes preguntas específicas:

- ¿Qué tan viable es ejecutar modelos de aprendizaje automático en diferentes placas Arduino y dispositivos de bajo costo?
- ¿Qué componentes debe integrar un agente inteligente embebido para operar de manera funcional dentro de un ecosistema IoT?
- ¿Cómo puede automatizarse la generación del agente inteligente reduciendo la complejidad técnica del desarrollo?
- ¿Qué tan interoperable, portable y eficiente resulta la solución propuesta en distintos escenarios y plataformas de hardware?

1.6 Objetivos

1.6.1 Objetivo general

Desarrollar una plataforma que genere agentes inteligentes para dispositivos Arduino, los cuales permitan analizar el comportamiento de los dispositivos en su entorno real de operación mediante modelos de inteligencia artificial, y realizar la integración de estos dispositivos al Internet de las Cosas (IoT).

1.6.2 Objetivos específicos

- OE1. Identificar las capacidades y limitaciones de cómputo de diferentes tipos de dispositivos Arduino a través de un estudio bibliográfico y experimental centrado en determinar las capacidades para soportar algoritmos de IA.

- OE2. Mejorar los enfoques actuales para entrenar modelos de IA desplegados en dispositivos Arduino utilizando los datos en tiempo real producidos por los sensores.
- OE3. Desarrollar un modelo de comunicación eficiente para el dispositivo Arduino que permita el envío directo de datos a plataformas de IoT.
- OE4. Desarrollar un enfoque en el que se entrenen los modelos de IA con datos históricos y se adapten para su ejecución embebida en dispositivos Arduino que realicen inferencias con datos obtenidos en tiempo real.
- OE5. Evaluar el desempeño del Agente Inteligente tomando en cuenta la eficiencia, confiabilidad y escalabilidad de la solución propuesta.

1.7 Metodología de solución

La metodología propuesta para este trabajo se estructura en fases secuenciales, cada una orientada a responder los objetivos planteados y a validar la hipótesis central. El enfoque combina actividades de revisión bibliográfica, diseño experimental, desarrollo tecnológico y pruebas en escenarios reales, garantizando la solidez científica y práctica del proyecto. El proceso metodológico se resume en cuatro bloques principales (Figura 1) que se explican a continuación.

Fase 1 – Evaluación de capacidades de cómputo

La primera fase de la metodología tiene como objetivo evaluar las placas de IoT de bajo costo más utilizadas en investigación y desarrollo para determinar su viabilidad como plataformas de ejecución de modelos de inteligencia artificial embebida. Para ello se seleccionaron cuatro placas representativas del ecosistema Arduino el Arduino Uno, el Nano 33 BLE, el ESP32 y el Portenta H7 Lite, abarcando un espectro de capacidades que va desde los 2 KB de RAM del Uno hasta el megabyte del Portenta. A lo largo de esta fase se identifican los tipos de sensores y datos que cada placa puede gestionar, se ejecutan pruebas preliminares para caracterizar los límites de memoria RAM, memoria Flash y velocidad de procesamiento de cada dispositivo, y se analiza el consumo energético bajo diferentes cargas computacionales. Los resultados de esta fase fundamentan directamente la Fase 2, pues determinan qué algoritmos de IA son desplegables en cada plataforma y qué optimizaciones son necesarias.

Fase 2 – Diseño del agente inteligente y modelo de comunicación

La segunda fase se centra en el diseño conceptual y matemático del agente inteligente embebido. Partiendo de los límites de hardware establecidos en la Fase 1, se seleccionan y optimizan los algoritmos de inteligencia artificial ligera (TinyML) Random Forest, Red Neuronal cuantizada y Árbol de Decisión que resultan viables para su ejecución en los dispositivos evaluados. En paralelo, se formaliza el agente mediante la función de decisión $f: P \times H \rightarrow A$, estableciendo con precisión el conjunto de percepciones capturadas por los sensores (P), el historial interno del agente representado en formato NGSI v2 (H) y el conjunto de acciones posibles condicionadas por el hardware (A). Esta fase incluye también la definición de los modelos de datos estandarizados para la interoperabilidad con FIWARE, el diseño de los

protocolos de comunicación (MQTT y HTTP) y la especificación de los módulos internos del agente para adquisición, inferencia, estandarización y transmisión de datos.

Fase 3 – Desarrollo e implementación

La tercera fase materializa el diseño elaborado en la Fase 2 mediante la construcción de la plataforma web generadora de agentes inteligentes. Esta plataforma, desarrollada en R con el framework Shiny, integra cinco módulos secuenciales: carga y visualización de datos históricos, entrenamiento y validación del modelo de IA con validación cruzada k=10, compresión TinyML mediante cuantización int8 y poda estructural, configuración de la comunicación IoT con FIWARE, y generación automática del firmware (.ino) listo para desplegar. Una vez generado, el agente se compila y carga en los dispositivos Arduino seleccionados en entornos controlados, verificando que el ciclo completo de percepción, inferencia, estandarización NGSI y transmisión a FIWARE funcione correctamente de extremo a extremo.

Fase 4 – Pruebas y análisis de resultados

La cuarta fase somete los agentes generados a una evaluación experimental rigurosa en dos casos de estudio contrastantes con relevancia práctica real. El primer caso de estudio aborda la detección de caídas en adultos mayores utilizando un Arduino Nano 33 BLE con el sensor inercial integrado LSM9DS1, evaluando la capacidad del agente para clasificar eventos de caída en tiempo real. El segundo caso aborda la estimación de evapotranspiración de referencia (ETo) en agricultura, donde el agente predice variables agrícolas a partir de lecturas de temperatura, humedad y velocidad del viento. En ambos casos se evalúa el rendimiento del agente considerando métricas de precisión predictiva (exactitud, F1, R^2, RMSE), eficiencia operativa (latencia, uso de RAM y Flash), confiabilidad (proporción de ciclos exitosos) y escalabilidad (modelo de colas M/M/1). Los resultados se contrastan con métodos tradicionales no embebibles para establecer el aporte diferenciador de la propuesta.

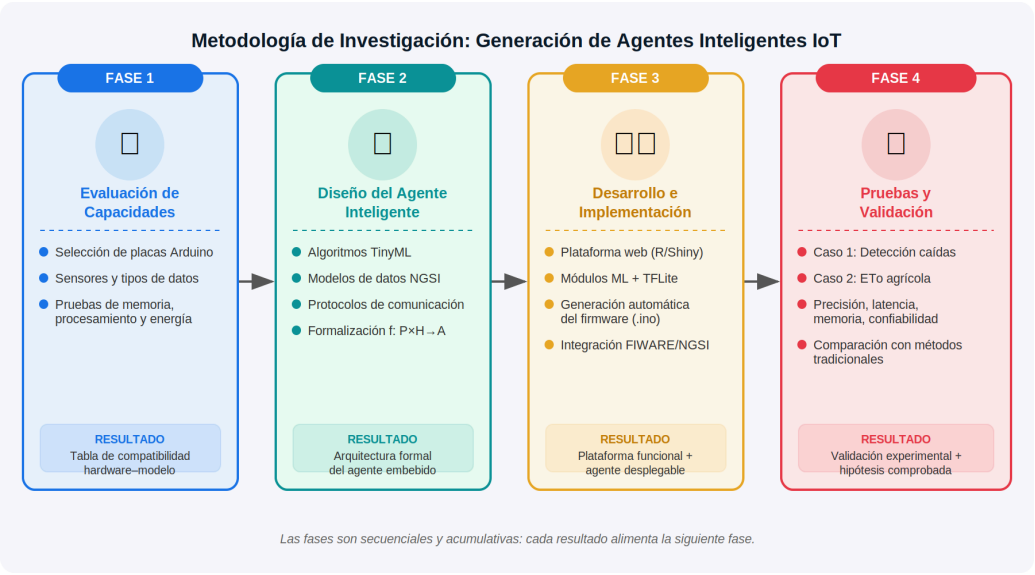

Figura 1: Metodología de un agente inteligente

Esta metodología asegura una transición clara desde la teoría y el diseño conceptual, hasta la validación práctica en escenarios reales, garantizando así la aplicabilidad de la propuesta.

1.8 Hipótesis

Esta investigación formula dos hipótesis complementarias, una centrada en el agente inteligente embebido y otra en la plataforma que lo genera:

Hipótesis 1 — Agente inteligente:

> *¿Es posible crear un agente inteligente de IoT capaz de desplegarse en un dispositivo Arduino que ejecute modelos de inteligencia artificial de forma local con los datos capturados en tiempo real por sus sensores, tome decisiones autónomas, estandarice la información capturada conforme al estándar NGSI v2, y la transmita a una plataforma IoT, manteniendo un desempeño funcional y eficiente a pesar de las restricciones computacionales del hardware?*

Hipótesis 2 — Plataforma de generación:

> *¿Es posible automatizar el proceso completo de creación de un agente inteligente de IoT desde el entrenamiento del modelo de IA hasta la generación del firmware desplegable en dispositivos Arduino a través de una plataforma software que no requiera programación manual del firmware por parte del usuario?*

1.9 Justificación

La presente investigación se justifica por razones científicas, tecnológicas, metodológicas y aplicadas.

Desde una perspectiva científica, esta investigación contribuye al avance del conocimiento en la intersección entre sistemas embebidos, aprendizaje automático y arquitecturas IoT. La propuesta formaliza el concepto de agente inteligente embebido mediante un modelo matemático explícito ($f: P \times H \rightarrow A$) que permite su análisis riguroso en términos de capacidad de percepción, restricciones temporales, confiabilidad operativa y escalabilidad. Esta formalización, aplicada a hardware de muy bajo costo, constituye una contribución original al campo de los sistemas multi-agente embebidos, ya que abre una línea de investigación reproducible y evaluable sobre la viabilidad de la inteligencia distribuida en el borde. Asimismo, la metodología PRISMA utilizada para la revisión sistemática y los protocolos de evaluación experimental desarrollados sientan precedentes metodológicos transferibles a trabajos futuros en Edge AI e IoT.

Desde una perspectiva tecnológica, existe una necesidad creciente de dotar a los dispositivos de bajo costo de mayores capacidades de autonomía y procesamiento local. En muchos contextos, depender exclusivamente de la nube para el análisis de datos resulta ineficiente debido a problemas de latencia, conectividad, consumo energético o privacidad. En este sentido, integrar modelos de inteligencia artificial directamente en el borde representa una alternativa relevante para construir soluciones más robustas y eficientes.

Desde una perspectiva metodológica, el desarrollo de soluciones IoT con inteligencia embebida sigue siendo un proceso complejo y poco estandarizado. La configuración manual de protocolos, modelos de datos, comunicación con plataformas y despliegue de modelos en hardware con capacidades limitadas de procesamiento representa una barrera importante para investigadores, desarrolladores y usuarios que buscan construir soluciones funcionales sin enfrentar una alta carga técnica. Por ello, una plataforma que automatice este proceso tiene un valor significativo en términos de accesibilidad, reproducibilidad y escalabilidad.

Desde una perspectiva aplicada, la propuesta tiene potencial de uso en múltiples dominios donde los dispositivos deben operar de forma autónoma en el entorno físico. Casos como la detección de caídas en adultos mayores, el monitoreo ambiental o la estimación de variables agrícolas muestran que un agente inteligente embebido puede aportar valor real al permitir una respuesta más inmediata, local y contextualizada.

Asimismo, esta investigación contribuye al campo de TinyML, Edge AI e IoT al no limitarse únicamente al despliegue aislado de modelos en microcontroladores, sino al integrar dichos modelos dentro de una arquitectura funcional de agentes inteligentes interoperables. Esto amplía el alcance de los dispositivos embebidos y fortalece su papel como nodos activos dentro de sistemas inteligentes distribuidos.

1.10 Aportes de la investigación

Los principales aportes de esta investigación se describen a continuación:

La presente investigación aporta una propuesta computacional orientada a fortalecer la convergencia entre inteligencia artificial embebida e Internet de las Cosas en dispositivos de bajos recursos. A diferencia de enfoques que se limitan únicamente a la conectividad de sensores o al despliegue aislado de modelos de aprendizaje automático, este trabajo plantea una solución integral centrada en la generación automática de agentes inteligentes embebidos capaces de operar de forma funcional dentro de un ecosistema IoT.

Uno de los principales aportes de esta investigación consiste en el diseño de una arquitectura modular que integra, dentro de un mismo agente, la adquisición de datos desde sensores, la ejecución local de modelos de inteligencia artificial, la estructuración contextual de la información y la comunicación interoperable con plataformas IoT. Esta arquitectura permite concebir al dispositivo no solo como un nodo de sensado o transmisión, sino como una entidad capaz de participar activamente en el análisis y procesamiento de información en el borde.

De manera complementaria, el trabajo aporta el desarrollo de una plataforma que automatiza la generación del agente inteligente, reduciendo la complejidad técnica asociada al entrenamiento de modelos, su adaptación a hardware con capacidades limitadas de procesamiento y su integración con plataformas como FIWARE. Esta automatización representa una contribución relevante, ya que transforma un proceso tradicionalmente manual y fragmentado en un flujo estructurado, reproducible y orientado a la interoperabilidad.

Finalmente, la investigación aporta evidencia experimental sobre la viabilidad de ejecutar inteligencia artificial embebida en microcontroladores de bajo costo, validando la propuesta en casos de estudio reales y en distintas placas de hardware. Esto permite demostrar que la solución no solo es conceptualmente sólida, sino también funcional en escenarios aplicados. En conjunto, estos elementos contribuyen al campo de TinyML, Edge AI e IoT al proponer una aproximación que trasciende la simple ejecución de modelos en microcontroladores y avanza hacia la construcción de agentes inteligentes embebidos interoperables.

1.11 Alcances y limitaciones

1.11.1 Alcances

La presente investigación se enfocó en el diseño, desarrollo e implementación de una plataforma orientada a la generación automática de agentes inteligentes embebidos en dispositivos de bajo costo, particularmente placas basadas en Arduino y microcontroladores afines.

El trabajo consideró como alcance principal la integración de tres dimensiones fundamentales: la adquisición de datos desde sensores físicos, la inferencia local mediante modelos de inteligencia artificial y la comunicación interoperable con plataformas del Internet de las Cosas.

Asimismo, la investigación contempló la evaluación experimental de la solución propuesta mediante casos de estudio reales y pruebas comparativas entre distintos modelos de inteligencia artificial y diferentes placas de hardware.

De manera específica, se validó el funcionamiento del agente inteligente en escenarios relacionados con la detección de caídas en adultos mayores y la estimación de evapotranspiración de referencia. Estos casos permitieron evaluar la viabilidad de la propuesta tanto en tareas de clasificación como en tareas de regresión.

Además, se consideró la integración con FIWARE como plataforma IoT principal, utilizando el estándar NGSI v2 para estructurar y transmitir la información generada por los agentes inteligentes.

1.11.2 Limitaciones

Entre las principales limitaciones del trabajo se encontraron las restricciones inherentes al hardware utilizado, tales como memoria RAM limitada, capacidad de procesamiento reducida y restricciones energéticas propias de los microcontroladores.

Además, la inteligencia incorporada en el agente estuvo orientada a tareas específicas de clasificación o predicción previamente definidas, por lo que no se abordó el desarrollo de inteligencia general ni de aprendizaje autónomo ilimitado en tiempo real.

De igual forma, aunque la propuesta contempló integración interoperable con plataformas IoT, el enfoque experimental se centró principalmente en FIWARE como entorno de validación. Por ello, la extensión hacia otras plataformas, como AWS IoT Core u otros entornos IoT, podría requerir adaptaciones adicionales.

Finalmente, la plataforma se orientó a un conjunto específico de modelos de aprendizaje automático y casos de estudio seleccionados. Por esta razón, la generalización hacia cualquier tipo de sensor, modelo o dominio de aplicación debe considerarse como una línea futura de expansión.

1.12 Organización del documento

El presente documento se organiza de la siguiente manera:

- En el Capítulo 2 se presenta el marco teórico que fundamenta el proyecto, incluyendo conceptos de IoT, agentes inteligentes, modelos de datos, protocolos de comunicación y fundamentos matemáticos.
- En el Capítulo 3 se expone el estado del arte, con una revisión sistemática de trabajos relacionados, análisis comparativo y vacíos de investigación.
- El Capítulo 4 describe al agente inteligente, fundamentos, funcionamiento, construcción e implementación.
- El Capítulo 5 describe la plataforma propuesta del sistema, su diseño modular y los componentes que integran al agente inteligente
- En el Capítulo 6 se muestran los resultados experimentales obtenidos a través de casos de estudio y pruebas de desempeño.
- En el Capítulo 7 se realiza la discusión de los resultados, destacando aportes, limitaciones y comparaciones con el estado del arte.
- Finalmente, en el Capítulo 8 se presentan las conclusiones generales y las líneas de trabajo futuro.

Capítulo 2 – Marco Teórico

El desarrollo de un agente inteligente para dispositivos de bajo costo, como Arduino, requiere un marco conceptual que integre de manera armónica los fundamentos del Internet de las Cosas (IoT), las plataformas de gestión de datos, los protocolos de comunicación, los modelos de datos estandarizados, los agentes inteligentes y los avances recientes en inteligencia artificial ligera. Además, se incluyen los fundamentos matemáticos que permiten formalizar el comportamiento del agente y evaluar su desempeño de manera rigurosa.

2.1 Internet de las Cosas (IoT)

El Internet de las Cosas (IoT, por sus siglas en inglés) término dado por Kevin Ashton en 1999 para describir un sistema donde objetos físicos estarían conectados a internet a través de sensores [2] ha transformado la forma en que los dispositivos interactúan con el entorno y con los seres humanos. Su propósito va más allá de la simple conexión: busca que los objetos físicos se conviertan en entes inteligentes, capaces de percibir, procesar y actuar en función del contexto.

Este paradigma se ha consolidado en aplicaciones como la automatización industrial, la salud, el transporte y la agricultura, donde los sensores generan grandes volúmenes de datos en tiempo real. Sin embargo, la diversidad de dispositivos y la falta de estandarización en protocolos de comunicación generan una alta complejidad en la integración. Así, el IoT se enfrenta a un dilema: la abundancia de datos frente a la dificultad para procesarlos y convertirlos en conocimiento útil de manera confiable y segura [1].

2.2 Plataformas IoT como entornos de integración

Para dar coherencia al ecosistema, se han creado plataformas IoT que actúan como intermediarias entre los dispositivos y las aplicaciones. Estas plataformas no solo almacenan información, sino que también proporcionan servicios para la gestión, el análisis y la visualización de datos [3].

2.2.1 FIWARE

FIWARE es una plataforma de código abierto orientada al desarrollo de aplicaciones inteligentes. Su núcleo es el Orion Context Broker, que gestiona información de contexto basada en el estándar NGSI. Gracias a este modelo, FIWARE permite que los datos recolectados se representen de forma uniforme y puedan ser reutilizados en diferentes aplicaciones. Su carácter abierto y modular la ha convertido en una referencia para proyectos de ciudades inteligentes y soluciones industriales [4].

Además de FIWARE, existen plataformas propietarias ampliamente adoptadas, como AWS IoT Core [5], que ofrecen servicios gestionados en la nube para la conexión, autenticación y procesamiento de datos de dispositivos IoT a gran escala. Si bien estas plataformas proveen mayor infraestructura de soporte, su carácter cerrado contrasta con el enfoque abierto e interoperable de FIWARE, razón por la cual esta investigación adopta FIWARE como entorno principal de validación.

2.3 Protocolos de comunicación en IoT

La comunicación es el pilar del IoT. Protocolos como MQTT, CoAP, HTTP y WebSockets determinan cómo se transmiten los datos entre dispositivos y plataformas [1][3]. Cada protocolo responde a diferentes necesidades de ancho de banda, latencia, consumo energético y fiabilidad de entrega, por lo que su selección constituye una decisión de diseño crítica en cualquier arquitectura IoT.

- MQTT es un protocolo ligero, especialmente diseñado para redes con ancho de banda limitado.
- CoAP se orienta a dispositivos embebidos con recursos muy restringidos.
- HTTP/HTTPS, aunque más pesado, es ampliamente utilizado en aplicaciones web e IoT.
- WebSockets permiten comunicación bidireccional en tiempo real.

La elección del protocolo no es trivial: condiciona la eficiencia energética, la latencia y la confiabilidad del sistema. En dispositivos como Arduino, donde cada byte de memoria y cada ciclo de reloj cuentan, seleccionar el protocolo adecuado es una decisión que impacta directamente en la viabilidad de la solución [6].

2.4 Modelos de datos en IoT

Transmitir datos no basta; es necesario darles estructura. Los modelos de datos son el lenguaje común que permite que diferentes sistemas comprendan y reutilicen la información generada por los dispositivos [3]. Sin un modelo de datos compartido, cada dispositivo genera información en su propio formato propietario, creando silos de datos que dificultan la integración e interoperabilidad en ecosistemas IoT distribuidos.

El estándar NGSI, utilizado en FIWARE, es un ejemplo de cómo los modelos de datos convierten la información en un recurso interoperable. Mediante entidades, atributos y metadatos, se logra que los datos se representen de forma clara, facilitando su análisis y posterior integración en diversas aplicaciones.

De esta forma, los modelos de datos son un factor crítico para evitar que los dispositivos IoT se conviertan en “islas de información” y, en cambio, formen parte de un ecosistema coherente y escalable [7].

2.5 Agente y Agentes inteligentes en IoT

2.5.1 Agentes

Un agente se define como una entidad autónoma capaz de percibir su entorno, procesar información y ejecutar acciones en función de objetivos específicos [8]. La noción de agente tiene raíces profundas en la inteligencia artificial clásica: Franklin y Graesser (1996) los caracterizaron como entidades que perciben el entorno mediante sensores y actúan sobre él mediante efectores, de forma continua y autónoma [30]. Más tarde, Jennings (2000) amplió esta definición al contexto de los sistemas de información, enfatizando que un agente genuino debe exhibir comportamiento reactivo (responder a cambios del entorno), comportamiento pro-activo (tomar iniciativa para alcanzar metas) y capacidad social (interactuar con otros agentes o sistemas) [31]. En el contexto de los sistemas multiagente, Wooldridge (2009) establece que los agentes poseen cuatro propiedades fundamentales: autonomía, capacidad social, reactividad y pro-actividad [8]. Estos atributos son los que distinguen a un agente de un simple programa: mientras un programa ejecuta instrucciones predefinidas, un agente interpreta su contexto y decide qué acción tomar en función de su estado interno y sus objetivos. Desde la perspectiva del aprendizaje automático, Russell y Norvig (2021) formalizan esta noción modelando al agente como una función de decisión [12]
$f: P \times H \rightarrow A$,, donde P representa las percepciones, H el historial interno, y A las acciones posibles. Esta abstracción permite entender a los agentes como sistemas racionales que actúan para maximizar su desempeño en un contexto determinado.
Los agentes pueden ser físicos (robots, sensores) o virtuales (programas de software), y su funcionamiento se basa en el ciclo percepción → decisión → acción. En aplicaciones tradicionales, los agentes se han utilizado para tareas como búsqueda en entornos complejos, planificación, y resolución de problemas distribuidos.

2.5.2 Agentes IoT

En el contexto del Internet de las Cosas, los agentes se adaptan para operar como mediadores entre dispositivos embebidos y plataformas de gestión de datos. La FIWARE Foundation [4] define un IoT Agent como un componente de software que actúa como intermediario entre dispositivos IoT y la plataforma de gestión de contexto, traduciendo los protocolos y formatos de datos del dispositivo al estándar NGSI v2 utilizado por el Orion Context Broker. Bajo esta definición, los agentes IoT abstraen la heterogeneidad de los dispositivos de campo y garantizan que la información llegue al ecosistema de gestión de datos en un formato uniforme e interoperable [7]. Se distinguen dos categorías principales:

- Agentes de dispositivo: responsables de recolectar datos directamente desde sensores, realizar preprocesamiento básico y garantizar la coherencia de la información.
- Agentes de plataforma: encargados de coordinar la transmisión, almacenamiento y análisis de datos en la nube, asegurando interoperabilidad y seguridad.

Los agentes IoT cumplen funciones críticas como:

- Transformar datos crudos en información estructurada mediante modelos estandarizados (ej. NGSI).

- Gestionar protocolos de comunicación (MQTT, CoAP, HTTP) para optimizar la latencia y consumo energético.
- Implementar mecanismos de seguridad que protejan la integridad y confidencialidad de los datos.

De esta forma, los agentes IoT permiten que dispositivos heterogéneos interactúen de manera coherente dentro de un ecosistema distribuido [3][4]. Esta arquitectura de mediación es precisamente la que esta investigación extiende al dotar al agente no solo de capacidades de transmisión, sino también de inferencia local mediante modelos TinyML.

2.5.3 Agentes inteligentes en IoT

Es importante aclarar que, en el contexto de este marco teórico, los agentes inteligentes en IoT no se presentan como sistemas ya consolidados o ampliamente disponibles en la literatura. Como se identificará en la revisión sistemática del Capítulo 3, la integración entre inferencia embebida (TinyML) y comunicación estandarizada con plataformas IoT sigue siendo una brecha abierta: los trabajos existentes abordan cada dimensión por separado pero no las integran en un agente completo y autónomo. En este sentido, el concepto que se presenta en esta sección constituye precisamente la dirección hacia la que apunta la propuesta de esta tesis. Se define, por tanto, un agente inteligente en IoT como la evolución conceptual de los agentes IoT tradicionales al dotarlos de capacidades de inteligencia artificial ligera. A diferencia de un agente IoT convencional que solo transmite datos crudos, un agente inteligente es capaz de reconocer patrones, ejecutar inferencias localmente y tomar decisiones autónomas sin depender de la nube para el procesamiento.

La viabilidad técnica de este concepto está habilitada por TinyML: la posibilidad de ejecutar modelos de clasificación y predicción en microcontroladores de bajo costo como Arduino, con latencias menores a 100 ms, hace que este tipo de agente sea implementable en hardware accesible. Esto abre la posibilidad de aplicaciones críticas como la detección de caídas, el monitoreo de signos vitales o la estimación de variables agrícolas. El Capítulo 4 de esta tesis propone y formaliza la arquitectura concreta de este agente inteligente embebido, cerrando la brecha identificada en la literatura.

En este sentido, los agentes inteligentes en IoT representan un cambio de paradigma: los dispositivos dejan de ser simples recolectores de datos y se convierten en nodos autónomos capaces de procesar, inferir y actuar localmente, incrementando la confiabilidad, la eficiencia y la escalabilidad de las soluciones IoT.

2.6 Inteligencia artificial ligera y TinyML en IoT

La Inteligencia Artificial ha demostrado su valor en el análisis de grandes volúmenes de datos, pero su integración en dispositivos de bajo costo ha sido históricamente limitada. La aparición de Tiny Machine Learning (TinyML) ha abierto nuevas posibilidades al permitir que modelos

previamente entrenados se optimicen para ejecutarse en microcontroladores con recursos muy restringidos [9].

Mediante técnicas como la poda, la cuantización y la reducción estructural [9][10], es posible reducir el tamaño de los modelos y adaptarlos para que funcionen en entornos con menos de 100 KB de memoria. Esto permite que placas como el Arduino Nano 33 BLE ejecuten inferencias en tiempo real con latencias menores a 100 milisegundos.

Este enfoque habilita un cambio de paradigma: en lugar de depender de la nube, los dispositivos IoT pueden procesar la información localmente, reduciendo la latencia, aumentando la privacidad y mejorando la eficiencia energética.

2.7 Formalización del agente inteligente como función de decisión

El agente no se define como un simple bloque de código, sino como un sistema racional autónomo, capaz de percibir, procesar y actuar. Modelar formalmente como una función permite definir con precisión su comportamiento y establecer límites operativos. f: P × H → A Este modelo, inspirado en la teoría de agentes de Russell y Norvig (2021) [12], facilita separar claramente las entradas (sensores), el estado interno (memoria o historial) y las acciones (decisiones) del sistema. Es especialmente relevante cuando se trabaja con restricciones embebidas, ya que obliga a formalizar el ciclo completo de percepción → decisión → acción [10]. Permite abstraer el diseño de software en términos funcionales, facilitar pruebas formales, y apoyar futuras extensiones del sistema a entornos multiagente o agentes adaptativos.

La IA embebida es el núcleo de la inteligencia del agente. Su comportamiento no es codificado por reglas fijas, sino aprendido a partir de datos. Por ello, los modelos de IA utilizados deben tener una formulación matemática explícita, que permita su entrenamiento, compresión y ejecución eficiente. $P(y \mid x) \approx f_\theta(x)$ Este enfoque permite usar redes neuronales ligeras o árboles de decisión como aproximadores de funciones sin perder interpretabilidad. Además, la formulación probabilística sustenta las métricas utilizadas (precisión, F1, AUC) y permite comparar versiones del modelo desde un enfoque cuantitativo [11][13]. En el contexto embebido, esta formulación ayuda a decidir cuál arquitectura puede mantener rendimiento sin exceder memoria o latencia, lo cual es vital para agentes desplegados en placas como ESP32 o Nano 33 BLE.

El agente opera en tiempo real. Cada ciclo de lectura, procesamiento y envío debe completarse antes de un límite temporal para que su decisión sea útil. Por eso se requiere una formulación explícita del tiempo total de ciclo: $T_{total} = T_{captura} + T_{procesamiento} + T_{comunicación} \leq T_{deadline}$ Esta ecuación guía directamente el diseño de la arquitectura del agente: cuánto puede tardar el modelo, cómo optimizar la comunicación, o si es necesario delegar alguna parte del procesamiento. Es crucial en contextos como detección de caídas, donde un retraso puede volver inútil la predicción [14].

Un sistema que 'funciona la mayor parte del tiempo' no es suficiente en aplicaciones reales. Es necesario tener una medición cuantitativa de la robustez del agente, especialmente en escenarios de operación prolongada, múltiples ciclos, y condiciones variables.R = N_ciclos_exitosos / N_ciclos_totales Además, puede aplicarse un intervalo de confianza binomial: IC_95% = R ± 1.96 * sqrt((R(1-R)) / N_ciclos_totales) Esto permite comparar versiones del agente y declarar formalmente si una versión supera a otra en condiciones similares [15]. Uno de los objetivos es que múltiples agentes puedan operar simultáneamente y enviar datos a una misma plataforma (FIWARE). Esto introduce un problema de escalabilidad: ¿el sistema sigue siendo eficiente cuando se multiplica el número de agentes? ρ = λ / μ Donde: - λ es la tasa de llegada de mensajes. - μ es la tasa de servicio del broker. Modelar el sistema como una cola tipo M/M/1 permite predecir cuellos de botella, estimar tiempos de espera y definir límites máximos de operación antes de saturar el canal o el broker. Es especialmente relevante en pruebas de escalabilidad horizontal [16]. Cabe destacar que esta sección introduce la formalización del agente como marco conceptual teórico. La concreción de cada uno de estos elementos qué conforman las percepciones P en términos de sensores reales, cómo se estructura el historial H mediante representaciones NGSI, y qué capacidades de inferencia determinan las acciones A según el hardware se desarrolla en detalle en el Capítulo 4, donde cada componente formal se aterriza en su implementación real sobre dispositivos Arduino. Los fundamentos teóricos presentados en este capítulo desde los conceptos del IoT y las plataformas de integración hasta la formalización matemática del agente inteligente embebido establecen el marco conceptual completo sobre el que se construye la revisión sistemática del Capítulo 3 y, posteriormente, el diseño de la solución propuesta.

Capítulo 3 – Estado del arte

3.1 Introducción y Metodología de Revisión

Este capítulo analiza el estado actual del conocimiento en tres áreas que convergen en esta investigación: la ejecución de inteligencia artificial ligera en microcontroladores (TinyML), la integración de dispositivos embebidos con plataformas IoT estandarizadas, y los enfoques que intentan combinar ambas dimensiones. El objetivo no es solo catalogar lo que existe, sino identificar con precisión las tres brechas que justifican la contribución de esta tesis.

Para garantizar una revisión exhaustiva, objetiva y reproducible, se aplicó una metodología basada en el enfoque PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) [17]. Este proceso estructura la búsqueda y selección de literatura de forma sistemática, reduciendo el sesgo de selección y asegurando que los trabajos incluidos sean relevantes para el problema de investigación.

3.1.1 Fuentes y estrategia de búsqueda

Las búsquedas se realizaron en seis bases de datos académicas: IEEE Xplore, ACM Digital Library, ScienceDirect, SpringerLink, Scopus y Web of Science. Se complementaron con documentos técnicos de proyectos de código abierto (FIWARE, TensorFlow Lite Micro, EdgeImpulse) y literatura. Las consultas se realizaron en inglés y español con operadores booleanos, usando cadenas como:

- **"Arduino AND Machine Learning"** / "TinyML on microcontrollers"
- **"IoT Agents AND FIWARE"** / "NGSI AND embedded systems"
- **"Edge computing AND embedded AI"** / "TinyML AND IoT integration"
- **"Automatic code generation AND microcontroller"** / "embedded agent AND IoT"

3.1.2 Criterios de inclusión y exclusión

Se incluyeron publicaciones entre 2018 y 2025 en revistas indexadas o conferencias internacionales, que reportaran implementaciones prácticas en hardware embebido de bajo costo (Arduino, ESP32, STM32, Portenta) o en plataformas IoT estandarizadas (FIWARE, AWS IoT, GCP). Se excluyeron artículos sin validación experimental, trabajos enfocados en hardware de alto rendimiento (servidores, GPUs), estudios duplicados y publicaciones sin acceso a texto completo.

3.1.3 Resultados del proceso de selección

La Tabla 1 presenta los resultados cuantitativos del proceso de filtrado aplicado sobre los registros identificados en las bases de datos consultadas.

Etapa del proceso PRISMA	Resultado
Registros identificados en bases de datos	120
Registros tras eliminar duplicados	98
Registros excluidos por criterios de inclusión/exclusión	58
Registros analizados en detalle	40
Trabajos finalmente incluidos en la revisión	**27**

Tabla 1. Resultados del proceso de selección PRISMA.

Los 27 trabajos incluidos se organizaron en tres brechas: trabajos de TinyML sin integración IoT (Brecha 1), trabajos de IoT sin IA embebida (Brecha 2), y trabajos con integración parcial que no cierran el ciclo completo (Brecha 3). Esta estructura muestra cómo la literatura ha avanzado por caminos paralelos y permite ubicar con precisión el aporte de esta investigación. De los 27 trabajos, se seleccionaron los 10 más representativos de cada brecha aquellos con mayor relevancia directa para los tres ejes de la contribución de esta tesis (IA embebida, integración IoT estándar y automatización) para presentarlos con análisis detallado en las secciones 3.2 a 3.4 y en la tabla comparativa (Tabla 2). Los 17 trabajos restantes corroboran los patrones identificados en cada brecha sin añadir dimensiones nuevas al análisis.

3.2 Brecha 1: Inteligencia Artificial Embebida sin Integración IoT

La primera línea de investigación demuestra que los microcontroladores de bajo costo pueden ejecutar modelos de aprendizaje automático localmente. Estos trabajos son fundamentales para establecer la viabilidad del paradigma TinyML y constituyen contribuciones sólidas dentro de su objetivo original: optimizar la inferencia embebida en hardware restringido. Es importante destacar que estos trabajos no fueron concebidos para integrarse con plataformas IoT, por lo

que señalar esa ausencia como una “desventaja” sería injusto con su intención. La distinción relevante para esta tesis no es que esos sistemas sean deficientes, sino que persiguen un objetivo diferente: ejecutar algoritmos de IA de forma eficiente en el dispositivo, sin que la transmisión estandarizada de resultados a sistemas externos sea parte de su alcance. Esta tesis, en cambio, parte de ese mismo punto de ejecución local y le añade una capa de integración IoT, cerrando un ciclo que esos trabajos, por diseño, dejaron abierto.

Warden y Situnayake (2019) establecieron los fundamentos del campo en su libro TinyML (O'Reilly Media), demostrando que modelos de reconocimiento de voz, gestos e imágenes podían ejecutarse en microcontroladores ARM Cortex-M mediante TensorFlow Lite Micro. Su trabajo define el marco técnico que los estudios posteriores extendieron. Sin embargo, el alcance se limita a la inferencia embebida: no contempla la comunicación estructurada con plataformas IoT ni la generación automática del firmware.

Izotov y Velichko (2021) llevaron TinyML a su límite al implementar el algoritmo LogNNet para reconocimiento de dígitos MNIST en un Arduino Uno de solo 2 KB de RAM, alcanzando un 82 % de precisión. El resultado es notable en eficiencia computacional, pero el sistema opera como clasificador aislado: no transmite predicciones a ninguna plataforma IoT ni usa formatos estándar. La contribución establece los límites inferiores de hardware viable sin abordar la dimensión de integración.

Viswanatha y Ramachandra (2022) implementaron modelos TinyML para reconocimiento de gestos y comandos de voz en un Arduino Nano 33 BLE con latencias inferiores a 100 ms. Su trabajo es el más cercano en hardware al de esta tesis, pero el ciclo termina en la inferencia local: las predicciones no se transmiten al broker IoT ni se encapsulan en formato NGSI. La ausencia de integración limita su aplicabilidad en ecosistemas IoT distribuidos.

Banbury et al. (2021) desarrollaron MLPerf Tiny, el primer benchmark estandarizado para sistemas TinyML, publicado en NeurIPS 2021. Establece métricas comparables de latencia, precisión y consumo energético, pero su alcance es el de un benchmark: no propone integración IoT ni generación automática de agentes.

Moin et al. (2022) presentaron en IEEE COMPSAC un enfoque de ingeniería dirigida por modelos para la generación automática de código TinyML orientado a IoT Edge. Es la propuesta más cercana en intención dentro de esta categoría: automatiza la generación del modelo embebido a partir de especificaciones de alto nivel. Sin embargo, la integración IoT es parcial no implementa NGSI v2 ni conectividad nativa con FIWARE y el artefacto generado es el modelo comprimido, no el firmware completo con lógica de comunicación.

Abadade et al. (2023) publicaron en IEEE Access una revisión exhaustiva del campo TinyML, analizando arquitecturas de optimización, técnicas de compresión (cuantización, poda, destilación de conocimiento) y dominios de aplicación en escenarios IoT distribuidos. Su trabajo establece el panorama técnico sobre el que se construyen las soluciones embebidas actuales, pero confirma que la integración estandarizada con plataformas IoT de middleware como FIWARE o AWS IoT Core permanece fuera del alcance de las propuestas revisadas [21].

Immonen e Hämäläinen (2022) presentaron un análisis detallado de TinyML para microcontroladores con recursos restringidos, examinando los límites de RAM, memoria de programa y velocidad de reloj, y evaluando técnicas de cuantización y poda para mantener precisión aceptable bajo dichas restricciones. Su trabajo describe múltiples frameworks TinyML y señala la dificultad de crear benchmarks comparables entre plataformas de hardware heterogéneo. La contribución queda circunscrita al ciclo de inferencia embebida sin considerar la transmisión estructurada de predicciones a plataformas IoT externas [22].

Hymel et al. (2022) presentaron Edge Impulse, una plataforma MLOps de extremo a extremo para TinyML que cubre adquisición de datos, entrenamiento, optimización y despliegue en múltiples microcontroladores. Su interfaz reduce significativamente la barrera técnica para desarrollar modelos embebidos. Sin embargo, la integración con middleware IoT requiere configuración manual por parte del desarrollador: no existe soporte nativo para NGSI ni interoperabilidad directa con ecosistemas como FIWARE, lo que limita su aplicabilidad en arquitecturas IoT estandarizadas [23].

Sudharsan et al. (2022) propusieron OTA-TinyML, un mecanismo de despliegue sobre el aire (over-the-air) de modelos TinyML en dispositivos IoT publicado en IEEE Internet Computing. Su propuesta extiende la automatización del ciclo de vida del modelo hacia la actualización remota de flotas de dispositivos heterogéneos, reduciendo la necesidad de intervención física. Pese a este avance en gestión de dispositivos, OTA-TinyML no incorpora módulos de comunicación estandarizada con plataformas de contexto IoT como FIWARE, ni genera agentes completos con lógica de inferencia y transmisión NGSI integrada [24].

Pavan et al. (2024) introdujeron TyBox, una caja de herramientas para el diseño automático y generación de código de modelos TinyML incrementales en el dispositivo, publicada en ACM Transactions on Embedded Computing Systems. A partir de un modelo TinyML estático, TyBox genera automáticamente su versión incremental adaptada a las restricciones de RAM del dispositivo y produce el código C++ necesario para ejecutar tanto inferencia como aprendizaje incremental directamente en el microcontrolador. Su evaluación en un Arduino Nano 33 BLE para clasificación de imágenes y reconocimiento de actividad humana demuestra la viabilidad del enfoque. Sin embargo, TyBox no incluye módulos para comunicación con protocolos IoT estandarizados (MQTT, HTTP, CoAP), ni para estructurar los datos según modelos de contexto como NGSI, ni para integrarse directamente con plataformas como FIWARE o AWS IoT Core, dejando incompleto el ciclo del agente inteligente IoT [25].

Patrón de la Brecha 1: los trabajos de esta categoría demuestran la viabilidad de ejecutar modelos de inteligencia artificial en microcontroladores de bajo costo, incluyendo tareas de reconocimiento, clasificación e inferencia embebida. Sin embargo, en la mayoría de los casos, la predicción generada por el modelo constituye el resultado final del proceso. En contraste, la propuesta de esta tesis utiliza modelos de IA como parte de un agente inteligente más amplio, capaz de operar con datos capturados en tiempo real por sensores, ejecutar inferencia local en Arduino, estructurar los resultados mediante NGSI v2 y comunicarlos a una plataforma IoT como FIWARE. Por ello, la brecha no se encuentra en la falta de calidad de los trabajos previos, sino en que su alcance se concentra en la inferencia embebida, mientras que esta investigación integra dicha inferencia dentro de un ciclo completo de agente inteligente IoT.

3.3 Brecha 2: Integración IoT Estandarizada sin Inteligencia en el Dispositivo

La segunda línea aborda la integración de dispositivos embebidos con plataformas IoT mediante estándares de datos. Estos trabajos construyeron ecosistemas de gestión de contexto robustos y ampliamente adoptados, pero mantienen al microcontrolador en un rol pasivo: captura y transmite datos crudos, mientras el procesamiento ocurre en la nube o en un gateway.

La FIWARE Foundation (2020) documentó el Orion Context Broker como componente central de su ecosistema abierto. El estándar NGSI v2 permite representar información de contexto de forma uniforme y reutilizable, adoptado en proyectos de ciudades inteligentes, agricultura de precisión y salud digital en más de 40 países. Su relevancia para esta investigación es directa: el agente propuesto adopta NGSI v2 como formato nativo. Sin embargo, la arquitectura de referencia de FIWARE contempla los dispositivos embebidos como fuentes de datos crudos gestionadas por IoT Agents en el servidor, no como entidades con capacidad de inferencia local.

Kamburugamuwe et al. (2021) presentaron en MDPI Sensors un sistema IoT consciente del contexto basado en FIWARE para entornos inteligentes, demostrando la viabilidad de integrar múltiples sensores heterogéneos con el Orion Context Broker mediante NGSI. El trabajo es referencia de integración IoT robusta, pero los microcontroladores actúan como nodos de adquisición sin IA local.

Nakamura et al. (2020) desarrollaron un IoT Agent para FIWARE compatible con el protocolo ECHONET Lite en dispositivos de automatización del hogar, presentado en IEEE GCCE 2020. Demuestra que FIWARE puede extenderse para protocolos propietarios mediante agentes de traducción en el gateway, pero el dispositivo de campo sigue siendo un sensor sin capacidades cognitivas.

Patrón de la Brecha 2: los trabajos construyen la infraestructura IoT que esta investigación aprovecha, pero definen al microcontrolador como nodo de adquisición. La inteligencia reside en el gateway o la nube. Esta separación entre el dispositivo que sensa y la entidad que decide es la brecha arquitectónica central que el agente inteligente propuesto busca cerrar.

3.4 Brecha 3: Intentos de Integración Parcial

La tercera categoría agrupa los trabajos más cercanos a esta propuesta: aquellos que combinan inferencia local con transmisión de datos, o que abordan las mismas aplicaciones evaluadas. Son los más relevantes para ubicar la contribución diferenciadora porque muestran exactamente hasta dónde ha llegado la literatura y dónde se detiene.

Daza Castillo (2023) implementó un sistema de detección de caídas en adultos mayores con Arduino Nano 33 BLE y sensor MPU6050, basado en reglas de umbral sobre la magnitud del vector de aceleración. Su trabajo es directamente comparable al Caso de Estudio 1 de esta tesis en hardware y el problema abordado. Sin embargo, el sistema utiliza un método de umbral

fijo sin aprendizaje automático dependiente de condiciones controladas de laboratorio y no implementa transmisión estandarizada de datos ni alertas. La ausencia de integración IoT limita su aplicabilidad en monitoreo remoto real.

Francisco Ruiz (2024) desarrolló modelos de estimación de evapotranspiración de referencia (ETo) mediante programación genética, directamente comparable al Caso de Estudio 2. Sus modelos logran R^2=0.931 y RMSE=0.189 ejecutándose en servidor con datos históricos completos. Sin embargo, la solución requiere infraestructura centralizada y no puede ejecutarse en un microcontrolador de campo: el modelo de programación genética no tiene representación compacta embebible en Arduino. No existe integración con plataformas IoT.

Moin et al. (2022), también presentes en la Brecha 1, representan el trabajo más cercano en automatización. Su plataforma genera modelos TinyML a partir de especificaciones de alto nivel, reduciendo el esfuerzo de ingeniería. Sin embargo, el artefacto generado es el modelo comprimido: el desarrollador debe integrar manualmente la lógica de comunicación IoT, el preprocesamiento de sensores y la encapsulación de datos. La ausencia de NGSI v2 y conectividad nativa con FIWARE limita su aplicabilidad industrial.

Yu et al. (2023) demostraron que una red neuronal convolucional compacta (TinyCNN) cuantizada puede ejecutarse en un Arduino Nano 33 BLE Sense para detección de caídas en tiempo real, alcanzando una sensibilidad superior al 99 % en dos conjuntos de datos públicos con una latencia de inferencia de 37 ms. El trabajo es directamente comparable al caso de estudio 1 de esta tesis en hardware y dominio de aplicación, y confirma la viabilidad técnica de los enfoques TinyML para monitoreo de caídas embebido. Sin embargo, la solución opera como clasificador autónomo: las predicciones no se transmiten a ninguna plataforma IoT estandarizada ni se encapsulan en formato NGSI, limitando su integración en sistemas de monitoreo remoto distribuido [26].

Hu et al. (2025) propusieron MicroFallNet, un modelo CNN ligero para detección de caídas en tiempo real desplegado en un ESP32 con sensores de muñeca. El modelo alcanza una media geométrica de precisión superior al 97 % con un tiempo de inferencia de 30 ms, destacando el potencial de la compresión de modelos para aplicaciones vestibles de bajo costo. Al igual que los trabajos anteriores de esta categoría, MicroFallNet opera de forma aislada: no contempla la comunicación estructurada con ecosistemas IoT ni el envío de predicciones a brokers de contexto bajo estándares como NGSI v2 [27].

Hu et al. (2022) presentaron un sistema de estimación de evapotranspiración de referencia (ETo) basado en IoT y aprendizaje automático, publicado en IEEE Access. El sistema emplea sensores de temperatura y humedad conectados a internet para adquirir variables meteorológicas y alimentar modelos ML entrenados en la nube para predecir el valor diario de ETo. Los resultados demuestran que la predicción precisa de ETo es alcanzable con variables reducidas mediante sensores IoT, lo que es coherente con el caso de estudio 2 de esta tesis. Sin embargo, la inferencia ocurre en la nube no en el dispositivo embebido y el sistema no genera agentes autónomos capaces de predecir localmente ni transmitir las predicciones en formato NGSI estandarizado [28].

Bashir et al. (2023) propusieron un framework de aprendizaje automático ensemble basado en LSTM para la predicción de ETo con datos climáticos adquiridos mediante sensores IoT, reportando un coeficiente de determinación R^2 superior a 0.97 bajo condiciones de variables reducidas. Su trabajo refuerza tanto la viabilidad del caso de uso de evapotranspiración con dispositivos IoT como la pertinencia de métricas RMSE y R^2 para evaluar modelos de regresión en este dominio. Sin embargo, al igual que Hu et al. (2022), la arquitectura centraliza el procesamiento en la nube sin contemplar la ejecución del modelo en el microcontrolador de campo ni la transmisión de predicciones mediante estándares NGSI [29].

Patrón de la Brecha 3: los trabajos abordan partes del problema inferencia embebida, automatización parcial o aplicaciones concretas pero ninguno integra simultáneamente: IA embebida en el dispositivo, comunicación estandarizada NGSI v2, y generación automática del agente completo desde una plataforma web.

3.5 Brecha Identificada y Posicionamiento de Esta Investigación

El análisis de los 27 trabajos revisados permite identificar con precisión la brecha que esta tesis aborda. Los tres grupos de literatura han avanzado por caminos paralelos sin convergencia:

- **Los trabajos de TinyML** demuestran que la inferencia embebida es viable en microcontroladores de bajo costo, pero evalúan el modelo como resultado terminal sin integración IoT.
- **Los trabajos de IoT con FIWARE** construyen infraestructuras de gestión de contexto robustas e interoperables, pero mantienen al microcontrolador como nodo pasivo de adquisición de datos.
- **Los intentos de integración parcial** abordan aplicaciones concretas o automatizan partes del flujo, pero ninguno cierra el ciclo completo desde la inferencia embebida hasta la transmisión estandarizada con generación automática del firmware.

La brecha central, no resuelta en la literatura revisada, es la siguiente: no existe una plataforma que genere automáticamente un agente inteligente completo —con modelo TinyML embebido, lógica de inferencia y comunicación NGSI v2 integrada— para dispositivos Arduino de bajo costo, sin requerir programación manual del firmware. Esta brecha tiene tres dimensiones simultáneas que ningún trabajo aborda de forma conjunta:

1. Dimensión de IA: los modelos TinyML deben ejecutarse directamente en el microcontrolador, no en la nube ni en un gateway.
2. Dimensión de IoT: los datos y predicciones deben transmitirse en formato NGSI v2 nativo, garantizando interoperabilidad con ecosistemas como FIWARE.
3. Dimensión de automatización: el ciclo completo desde el entrenamiento hasta el firmware desplegable debe ser accesible sin conocimientos avanzados de programación embebida.

Esta tesis propone cerrar esa brecha mediante una plataforma web que integra las tres dimensiones en un flujo automatizado, generando agentes inteligentes embebidos validados en escenarios reales de salud y agricultura. La Tabla 2 sintetiza la posición de esta investigación respecto al estado del arte:

Trabajo	Enfoque	IA embebida	Integración IoT / NGSI	Generación automática	Brecha
Warden & Situnayake (2019)	TinyML: fundamentos y despliegue en microcontroladores ARM.	Sí — base TFLite Micro.	No — sin integración IoT.	No.	Sin IoT.
Izotov & Velichko (2021)	LogNNet MNIST en Arduino Uno, 82 % precisión en 2 KB RAM.	Sí — modelo en 2 KB.	No — inferencia aislada.	No.	Sin IoT.
Viswanatha & Ramachandra (2022)	TinyML gestos/voz en Nano 33 BLE. Latencia <100 ms.	Sí — TFLite Micro cuantizado.	No — sin transmisión estándar.	No — firmware manual.	Sin IoT.
Banbury et al. (2021)	MLPerf Tiny: benchmark de inferencia TinyML en MCUs.	Sí — evalúa latencia/precisión.	No — benchmark sin IoT.	No.	Sin IoT.

Trabajo	Enfoque	IA embebida	Integración IoT / NGSI	Generación automática	Brecha
Moin et al. (2022)	Generación automática de modelos TinyML para IoT Edge.	Sí — genera modelos embebidos.	Parcial — sin FIWARE/NGSI nativo.	Parcial — modelo, no firmware completo.	Sin NGSI v2 completo.
Kamburugamuwe et al. (2021)	Sistema IoT context-aware con FIWARE para entornos inteligentes.	No — datos crudos sin IA local.	Sí — FIWARE + NGSI.	No.	Sin IA en dispositivo.
Nakamura et al. (2020)	IoT Agent FIWARE para protocolo ECHONET Lite en smart home.	No — agente de traducción.	Sí — FIWARE + NGSI v2.	No.	MCU como nodo pasivo.
Daza Castillo (2023)	Detección de caídas con umbrales en Nano 33 BLE + MPU6050.	No — reglas fijas sin ML.	No — sin transmisión estándar.	No.	Sin IA ni IoT.

Trabajo	Enfoque	IA embebida	Integración IoT / NGSI	Generación automática	Brecha
Francisco Ruiz (2024)	Estimación ETo con programación genética. R^2=0.931.	No — ejecución en servidor.	No — sin integración IoT.	No.	Modelo no embebible.
Esta tesis	Plataforma web genera agentes TinyML completos para Arduino con FIWARE.	Sí — RF/NN/DT int8 (45–65 KB).	Sí — NGSI v2, FIWARE, CrateDB, Grafana.	Sí — sketch .ino automático.	—

Tabla 2. Comparación entre trabajos representativos del estado del arte y la propuesta de esta tesis.

La tabla confirma que esta tesis es la única propuesta que cubre simultáneamente las tres dimensiones de la brecha identificada: IA embebida funcional en hardware de bajo costo, comunicación estandarizada NGSI v2 con FIWARE, y generación automática del agente completo. Esta convergencia constituye la aportación diferenciada de la investigación respecto al estado del arte, y establece el punto de partida desde el que se desarrollan los capítulos siguientes. Con la brecha identificada con precisión, el Capítulo 4 presenta el modelo conceptual y la formalización matemática del agente inteligente embebido propuesto para cerrarla.

Capítulo 4 – Agentes inteligentes IoT propuestos

4.1 Introducción

El desarrollo de agentes inteligentes embebidos constituye el eje central de esta investigación. A diferencia de dispositivos IoT tradicionales, que se limitan a la adquisición y transmisión de datos, el agente propuesto integra capacidades de percepción, decisión y comunicación estructurada directamente en el dispositivo. Esta aproximación permite trasladar procesos de

inferencia al borde de la red, reduciendo la dependencia de la nube y mejorando la autonomía del sistema.
El agente no se concibe como un simple programa, sino como un sistema racional autónomo, capaz de percibir, procesar y actuar en tiempo real. Su diseño modular y su formalización matemática permiten evaluar su desempeño de manera rigurosa, garantizando confiabilidad y escalabilidad.
Es importante destacar que la selección de placas Arduino y algoritmos de aprendizaje automático no fue arbitraria. Se realizó un estudio exhaustivo de la literatura y de las capacidades de diferentes placas (Arduino Uno, Nano 33 BLE, Portenta H7 Lite y Wemos D1 ESP8266), analizando limitaciones de memoria, procesamiento y consumo energético. Este análisis permitió identificar qué dispositivos eran viables para ejecutar modelos TinyML y qué optimizaciones eran necesarias para lograr un desempeño aceptable en condiciones reales. Durante el estudio se encontró que Arduino Uno, Portenta H7 Lite, Nano 33 BLE y ESP32 eran las placas más adecuadas para trabajar con modelos ML. Sin embargo, no todas soportan los mismos modelos ni la misma cantidad de datos; por ello, la optimización de los modelos permite que placas de menor potencia ejecuten modelos ML hasta cierto límite operativo [9,10].
Este capítulo describe el modelo conceptual del agente inteligente embebido desarrollado en esta tesis. Se presenta su arquitectura funcional, su formalización matemática, su ciclo operativo y su integración conceptual con dispositivos Arduino. El objetivo es establecer una definición precisa del agente antes de describir la plataforma que automatiza su generación.

4.2 Arquitectura funcional del agente

Un agente inteligente embebido, en el contexto de esta investigación, se define como una entidad de software que opera dentro de un microcontrolador, capaz de:

- Percibir variables del entorno mediante sensores físicos.
- Ejecutar un modelo de aprendizaje automático previamente entrenado.
- Estandarizar los resultados en un modelo de datos interoperable.
- Comunicar dicha información a una plataforma IoT.

A diferencia de un agente IoT tradicional, cuya función principal es transmitir datos, el agente propuesto incorpora inferencia local. Esto implica que la toma de decisiones no ocurre en la nube, sino directamente en el dispositivo, respetando restricciones de memoria, procesamiento y tiempo de ejecución. El agente no realiza entrenamiento en línea ni adaptación dinámica del modelo. Su comportamiento se basa en la ejecución determinista de un modelo previamente validado por la plataforma de generación. Esta decisión de diseño garantiza estabilidad, trazabilidad y control experimental.

La Figura 2 presenta la arquitectura funcional del agente inteligente propuesto. Esta arquitectura organiza el comportamiento del agente en cuatro módulos principales: adquisición de datos, inferencia local, estandarización de la información y comunicación IoT. En primer lugar, el agente obtiene datos del entorno mediante sensores conectados al dispositivo Arduino.

Posteriormente, estos datos son procesados por un modelo de inteligencia artificial embebido, encargado de generar una clasificación o predicción de acuerdo con el caso de estudio.

Una vez realizada la inferencia, el agente estructura los datos sensados y el resultado del modelo mediante el estándar NGSI v2, lo que permite representar la información como una entidad de contexto interoperable. Finalmente, el módulo de comunicación transmite dicha entidad hacia FIWARE, permitiendo su almacenamiento, consulta y graficación posterior. De esta manera, la arquitectura permite que el dispositivo Arduino no funcione únicamente como un nodo de sensado, sino como un agente inteligente capaz de percibir, analizar y comunicar información contextualizada dentro de un ecosistema IoT.

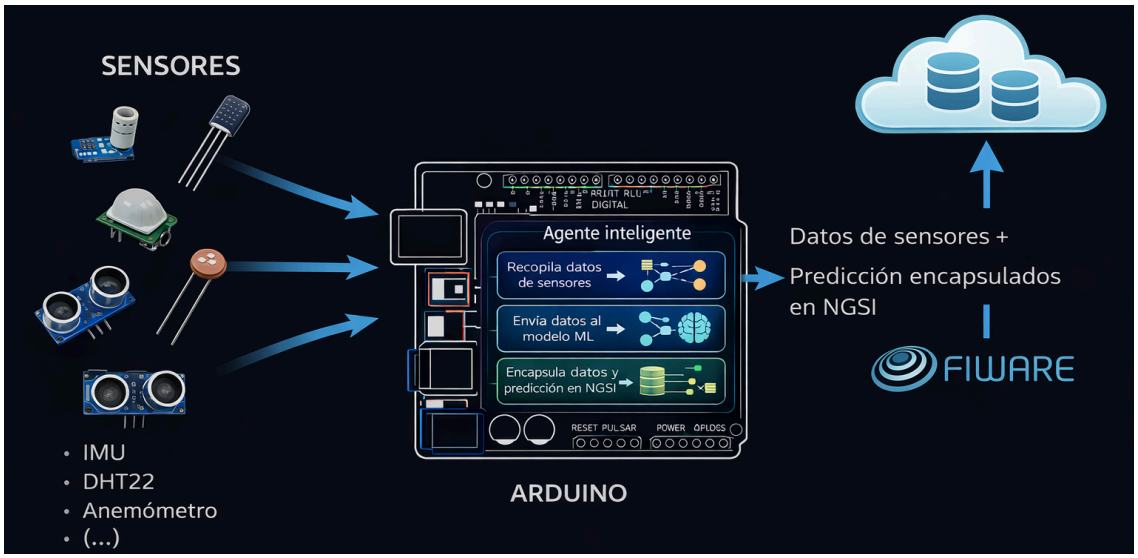

Figura 2: Arquitectura del agente inteligente

4.3 Formalización matemática del agente

El agente no se define como un simple bloque de código, sino como un sistema racional autónomo, capaz de percibir, procesar y actuar. Modelarlo formalmente como una función permite definir con precisión su comportamiento y establecer límites operativos:

$f: P \times H \rightarrow A$

P: conjunto de percepciones capturadas por los sensores. En la implementación concreta del agente, P está compuesto por el conjunto de lecturas simultáneas de los sensores físicos conectados al dispositivo. El número y tipo de sensores que puede gestionar el agente depende directamente del tipo de placa: el Arduino Uno soporta hasta 6 entradas analógicas (0–5 V) y 14 pines digitales, adecuados para sensores básicos de temperatura, humedad o presencia; el Nano 33 BLE integra sensores IMU (acelerómetro, giroscopio, magnetómetro), temperatura y micrófono directamente en la placa, además de buses I²C y SPI para sensores externos; el ESP32 dispone de 18 canales ADC de 12 bits con WiFi y Bluetooth nativos; y el

Portenta H7 Lite soporta conectividad industrial y alta densidad de sensores con su arquitectura dual-core. En todos los casos, los datos sensoriales son valores numéricos que el agente organiza en el vector xt, construido en memoria como un arreglo de valores preprocesados listos para la inferencia. Cada lectura puede ser analógica (voltaje convertido a valor digital) o digital (protocolo I²C, SPI o UART), y el agente valida cada valor antes de incorporarlo al vector para prevenir que datos erróneos lleguen al modelo.

H: historial interno del agente (memoria, estados previos). En la implementación sobre Arduino, H se materializa a través de la representación NGSI v2: en cada ciclo, el agente construye una entidad JSON conforme a este estándar que encapsula los valores sensoriales actuales, la predicción generada por el modelo TinyML y metadatos esenciales (identificador de entidad, tipo de entidad y marca temporal). NGSI v2 (Next Generation Service Interface) es el estándar de modelos de datos abiertos adoptado por FIWARE, que define cómo representar información de contexto IoT de forma uniforme e interoperable. Una entidad NGSI tiene la forma: {"id": "urn:ngsi-ld:Device:001", "type": "IoTAgent", "accel_x": {"value": 1.23, "type": "Float"}, "prediccion": {"value": 1, "type": "Integer"}, "timestamp": {"value": "2025-01-01T00:00:00Z", "type": "DateTime"}}. Esta representación permite que el estado del agente no solo sea procesado localmente, sino transmitido al Orion Context Broker de FIWARE y almacenado históricamente en CrateDB para su posterior análisis y visualización en Grafana. Así, H no es solo memoria local efímera: es estado contextual interoperable dentro del ecosistema IoT.

A: conjunto de acciones posibles (ejecución de inferencias, transmisión de datos, registro de fallos). La capacidad real de A en un agente embebido está directamente condicionada por la arquitectura de hardware de la placa. El estudio experimental realizado en esta investigación (Capítulo 6) establece con precisión estos límites: el Arduino Uno (ATmega328P, 2 KB RAM, 32 KB Flash) solo puede ejecutar modelos de tipo Árbol de Decisión reducidos (≤45 KB), con una latencia de 160 ms, y opera al límite de sus recursos con un 90 % de RAM ocupada; el Nano 33 BLE (ARM Cortex-M4, 256 KB RAM, 1 MB Flash) soporta Random Forest, Red Neuronal cuantizada y Árbol de Decisión (45–65 KB) con latencias de 35 ms y uso moderado de memoria; el ESP32 (dual-core Xtensa, 520 KB RAM, 4 MB Flash) ejecuta los mismos modelos con 25 ms de latencia y mayor margen de memoria; y el Portenta H7 Lite (dual-core Cortex-M7/M4, 1 MB RAM, 2 MB Flash) admite todos los modelos evaluados con latencias de apenas 10 ms. Por tanto, A no es un conjunto abstracto: está delimitado por cuánta memoria Flash puede almacenar el modelo TinyML, cuánta RAM requiere su ejecución y si el procesador puede completar la inferencia dentro del Tdeadline requerido por la aplicación. Esta dependencia hardware-acción es una característica definitoria del agente inteligente embebido que lo distingue fundamentalmente de un agente de software convencional.

Ejemplo concreto: agente de detección de caídas en Arduino Nano 33 BLE

Para ilustrar la función f: P × H → A con un dispositivo concreto, considérese el agente de detección de caídas en adultos mayores implementado sobre un Arduino Nano 33 BLE (Caso de Estudio 1, Capítulo 6). En este agente: P está compuesto por las lecturas del sensor inercial IMU LSM9DS1 integrado en la placa, que captura aceleración en tres ejes y velocidad angular

en tres ejes, formando un vector x□ = [a_x, a_y, a□, g_x, g_y, g□] de 6 valores float muestreados en cada ciclo. H se materializa como una entidad NGSI v2 que incluye los seis valores sensoriales, la predicción del modelo (0 = actividad normal, 1 = caída detectada) y una marca temporal, todo encapsulado en un objeto JSON transmitido al Orion Context Broker de FIWARE. A incluye tres acciones posibles: (1) transmitir la entidad NGSI al broker mediante HTTP POST si la inferencia y el tiempo son válidos, (2) registrar el ciclo como exitoso, o (3) registrar el fallo específico (percepción, inferencia o comunicación) si algo falla. El agente ejecuta este ciclo con una latencia total de 25–35 ms, bien por debajo del Tdeadline definido para la detección de caídas, y alcanza una confiabilidad R ≥ 0.9994 en sesiones de 24 horas.
Este modelo, inspirado en la teoría de agentes de Russell y Norvig (2021) [12], facilita separar claramente las entradas, el estado interno y las acciones del sistema. Es especialmente relevante en sistemas embebidos, ya que obliga a formalizar el ciclo completo de percepción → decisión → acción [10].

Inteligencia artificial embebida
La IA embebida es el núcleo del agente. Su comportamiento no está codificado por reglas fijas, sino aprendido a partir de datos. Por ello, los modelos de IA utilizados deben tener una formulación matemática explícita:

$$P(y|x) \approx f\theta(x)$$

x: vector de entrada con datos sensoriales preprocesados.

y: salida esperada (clase, predicción o valor estimado).

$f\theta(x)$: función aproximadora parametrizada por θ (pesos del modelo).

θ: parámetros del modelo entrenado (red neuronal ligera, árbol de decisión, etc.).

Este enfoque permite usar modelos ligeros como redes neuronales compactas o árboles de decisión, manteniendo interpretabilidad y eficiencia.

Restricciones temporales
El agente opera en tiempo real, lo que exige una restricción explícita:

$$Ttotal = Tcaptura + Tprocesamiento + Tcomunicacio´n \leq Tdeadline$$

$Tcaptura$: tiempo de adquisición de datos desde los sensores.

$Tprocesamiento$: tiempo requerido para preprocesar datos y ejecutar el modelo TinyML.

$Tcomunicación$: tiempo de construcción y transmisión del mensaje IoT.

$Tdeadline$: límite máximo permitido para que la decisión sea útil en el contexto de aplicación.

Confiabilidad
La confiabilidad del agente se define como:

$R = Nciclos\ exitosos/Nciclos\ totales$
$Nciclos\ exitosos$: número de ciclos completados sin fallos.

$Nciclos\ totales$: número total de ciclos ejecutados.

R: proporción de éxito, indicador de robustez del agente.

Para evaluar la robustez estadística, se utiliza un intervalo de confianza binomial:

$IC95\% = R \pm 1.96 \cdot R(1 - R)Nciclos\ totales$

Esto permite comparar versiones del agente y declarar formalmente si una configuración supera a otra en condiciones similares.

Escalabilidad
Uno de los objetivos es que múltiples agentes puedan operar simultáneamente y enviar datos a una misma plataforma (FIWARE). Esto introduce un problema de escalabilidad, modelado mediante teoría de colas:

$\rho = \lambda\mu$

λ: tasa de llegada de mensajes generados por los agentes.

μ: tasa de servicio del broker o plataforma IoT.

ϱ: factor de utilización del sistema, indicador de saturación.

Modelar el sistema como una cola tipo M/M/1 permite predecir cuellos de botella, estimar tiempos de espera y definir límites máximos de operación antes de saturar el canal o el broker.Por ejemplo, si se despliegan 10 agentes sobre Arduino Nano 33 BLE que envían mensajes NGSI cada 35 ms (λ ≈ 28.6 mensajes/s por agente, es decir λtotal ≈ 286 mensajes/s en total), y el Orion Context Broker puede atender μ = 600 solicitudes/s, entonces ρ = 286/600 ≈ 0.48. Un factor de utilización del 48 % indica que el sistema opera con margen suficiente. Si se añadieran 20 agentes adicionales (λtotal ≈ 858 mensajes/s), ρ superaría 1.0 y el sistema entraría en saturación, lo que justifica dimensionar la plataforma IoT antes del despliegue masivo.

4.4 Ciclo operativo del agente

El agente opera de manera continua mientras el dispositivo permanezca activo. Cada ciclo inicia registrando el instante inicial de tiempo y ejecuta las siguientes etapas.

Primero, se realiza la lectura de sensores y la validación del vector xt Si la lectura no es válida, el sistema registra un fallo y pasa al siguiente ciclo sin ejecutar el modelo.

Si la lectura es válida, el modelo ML embebido se ejecuta mediante propagación hacia adelante. Internamente, el vector de entrada es copiado al buffer del modelo y se realiza el cálculo correspondiente. En modelos de clasificación, la salida corresponde a probabilidades asociadas a clases; el agente selecciona la clase de mayor probabilidad como predicción final. En modelos de regresión, la salida es un valor continuo estimado.

El resultado $y\hat{}t$se encapsula posteriormente en una estructura JSON compatible con NGSI. El mensaje incluye identificador de entidad, atributos sensados, predicción generada y marca temporal.

Tras construir el mensaje, el agente verifica que el tiempo total del ciclo no exceda el valor definido como $Tdeadline$. Si el tiempo excede el límite, el ciclo se marca como fallo por restricción temporal.

En caso contrario, el mensaje se transmite a la plataforma IoT mediante el protocolo configurado. Si la transmisión falla, se registra el evento como fallo de comunicación.

Este diseño asegura que ante cualquier fallo lectura inválida, error en ejecución del modelo, error de estandarización, fallo de comunicación o exceso de tiempo el agente no se detiene, sino que continúa operando en el siguiente ciclo, permitiendo evaluar su robustez bajo condiciones reales.

La Figura 3 presenta el diagrama de flujo completo del ciclo operativo implementado en el firmware del dispositivo.

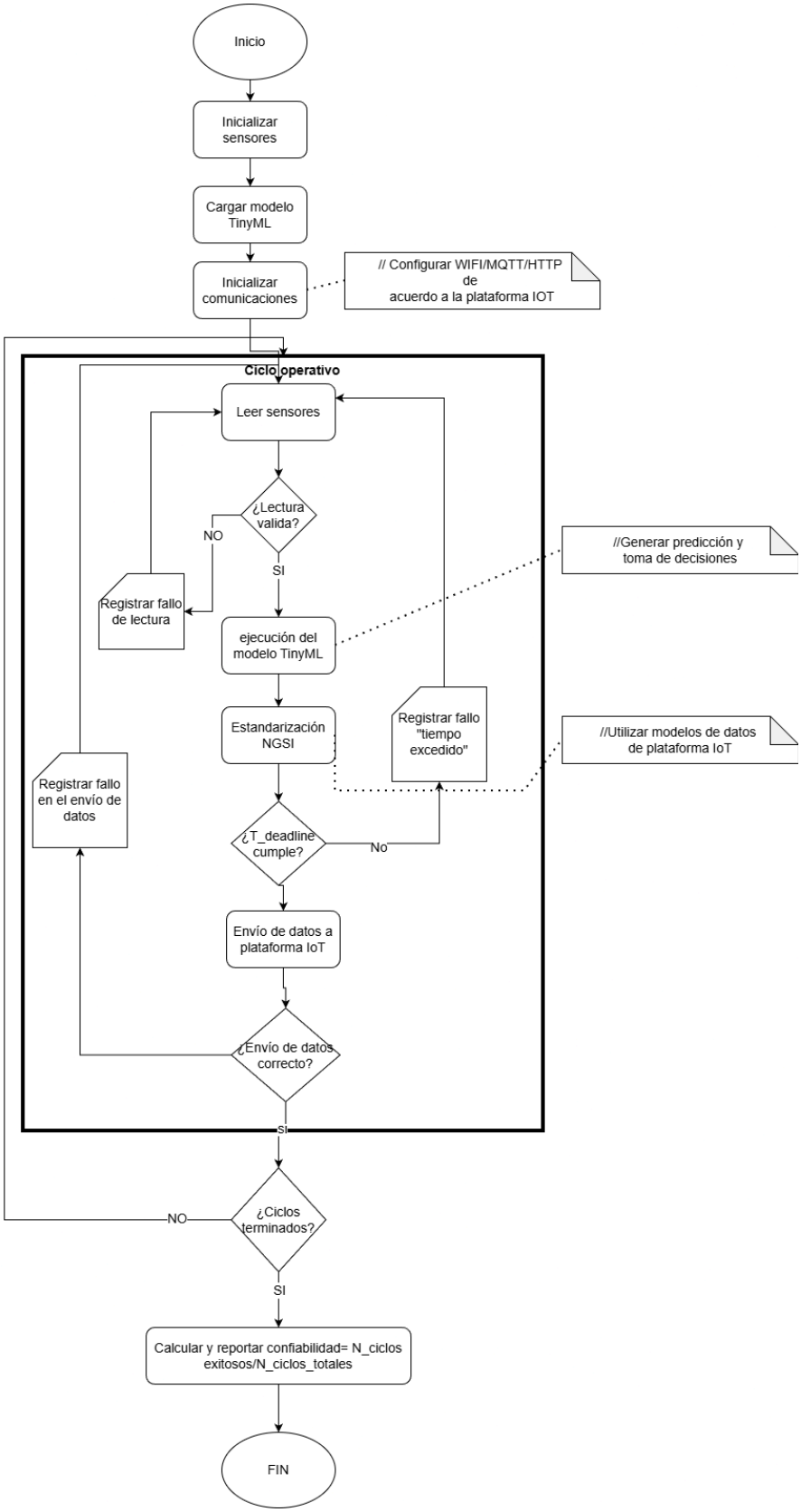
Inicio
Inicializar sensores
Cargar modelo TinyML
Inicializar comunicaciones
// Configurar WIFI/MQTT/HTTP de acuerdo a la plataforma IOT
Ciclo operativo
Leer sensores
¿Lectura valida?
NO
SI
Registrar fallo de lectura
ejecución del modelo TinyML
//Generar predicción y toma de decisiones
Estandarización NGSI
Registrar fallo "tiempo excedido"
//Utilizar modelos de datos de plataforma IoT
Registrar fallo en el envío de datos
¿T_deadline cumple?
No
Envío de datos a plataforma IoT
¿Envío de datos correcto?
SI
¿Ciclos terminados?
NO
SI
Calcular y reportar confiabilidad= N_ciclos exitosos/N_ciclos_totales
FIN

Figura 3: Diagrama de flujo del agente inteligente

4.5 Integración del Agente en dispositivos Arduino

La integración del agente inteligente en dispositivos Arduino constituye un componente esencial de esta tesis, ya que materializa el modelo conceptual presentado en las secciones previas dentro de un entorno embebido con restricciones severas de memoria, cómputo y energía. En este trabajo, el agente no se concibe como un prototipo teórico, sino como un firmware ejecutable capaz de operar de forma continua, leer sensores, ejecutar un modelo de aprendizaje automático embebido, estandarizar la información y transmitirla a una plataforma IoT. Esta sección organiza la descripción de la implementación siguiendo el mismo orden de la formalización matemática del agente definida en la sección 4.3: primero el conjunto de percepciones P (adquisición de sensores), luego el historial interno H (inferencia y representación NGSI), después el conjunto de acciones A (comunicación IoT), y finalmente el control temporal y de confiabilidad.

Una primera consideración crítica es que el microcontrolador no dispone de los recursos típicos de un entorno de cómputo general. La memoria RAM, la memoria Flash y la capacidad de procesamiento son limitadas, y esto condiciona la forma en que se representan los datos, se almacena el modelo y se ejecuta la inferencia. Por ello, el agente fue diseñado bajo un principio de ejecución determinista y modular.

Como referencia para las subsecciones que siguen, la función del agente se define formalmente como:

$$f : P \times H \rightarrow A$$

donde P es el conjunto de percepciones, H el historial interno (representado en NGSI v2) y A el conjunto de acciones posibles. Cada subsección describe cómo uno de estos componentes se concreta en el firmware del dispositivo Arduino.

Implementación de P: adquisición y validación de percepciones

En la implementación concreta sobre un dispositivo Arduino, el conjunto P el conjunto de percepciones del agente está integrado por todos los sensores físicos conectados al dispositivo IoT donde se desplegará el agente inteligente. En cada ciclo de operación, P se construye mediante la lectura secuencial de cada uno de esos sensores configurados en el dispositivo de destino: el agente lee sensor por sensor y acumula sus valores hasta tener el conjunto completo de percepciones del ciclo actual.

El tipo de lectura depende de la interfaz eléctrica del sensor. Las entradas analógicas, es decir, los sensores que entregan una señal de voltaje continuo, como termistores o sensores de luz son leídas mediante la función analogRead() de Arduino, que convierte ese voltaje en un entero de 10 bits con rango 0–1023, proporcional al voltaje medido entre 0 V y la tensión de referencia de la placa. Los sensores digitales con protocolos de bus como el sensor inercial IMU LSM9DS1 integrado en el Nano 33 BLE, que comunica aceleración, velocidad angular y campo

magnético mediante I²C son interrogados mediante bibliotecas especializadas que entregan directamente los valores físicos calibrados en unidades del dominio (m/s², °/s, μT).

El resultado de todas estas lecturas es el vector de características xt, que es precisamente la representación del conjunto P para el ciclo t. El vector xt es un arreglo de valores numéricos de tipo float almacenado en memoria RAM del microcontrolador, donde cada posición corresponde a una variable sensorial específica (por ejemplo, posición 0 = aceleración en eje X, posición 1 = aceleración en eje Y, etc.). El tamaño del vector es decir, cuántas variables incluye está determinado por el número de variables de entrada con las que fue entrenado el modelo TinyML embebido en ese agente.

Antes de incorporar cada dato obtenido de los sensores al vector xt, el agente aplica validaciones básicas a cada dato obtenido de los sensores: verifica que el valor no sea nulo (lectura fallida), que esté dentro del rango físico razonablemente esperado para ese sensor (por ejemplo, que una temperatura no sea -200 °C ni 500 °C), y que la comunicación con el sensor no haya generado un código de error en el bus I²C o SPI. Estas validaciones tienen un propósito técnico fundamental: evitar que datos erróneos entren al modelo TinyML, ya que un valor fuera de rango puede desbordar buffers numéricos o producir predicciones completamente inconsistentes. Si la validación de cualquier dato falla, el ciclo completo se registra como fallo de percepción y el agente avanza directamente al siguiente ciclo sin ejecutar la inferencia, preservando así la estabilidad operativa del firmware.

Implementación de H: inferencia embebida y representación NGSI

Una vez construido el vector xt, el agente prepara la inferencia normalizando los valores según los parámetros del modelo y los carga en el buffer de entrada del motor TinyML. El modelo queda enlazado al firmware como un arreglo de bytes constante en Flash no se descarga dinámicamente y la inferencia se implementa como una llamada directa a la rutina del modelo, que recibe xt y retorna ŷt. En clasificación, ŷt es la clase de mayor probabilidad; en regresión, un valor continuo. El agente verifica la consistencia mínima de ŷt como mecanismo de seguridad embebida.

El historial H se materializa mediante la construcción de la entidad NGSI v2: los valores de xt y ŷt se empaquetan junto con el identificador de entidad, el tipo de entidad y la marca temporal en un objeto JSON directamente consumible por el Orion Context Broker de FIWARE. El modelo de datos NGSI garantiza interoperabilidad: la misma entidad generada por un agente sobre un Nano 33 BLE es legible por cualquier componente del ecosistema FIWARE, desde QuantumLeap para series temporales hasta Grafana para visualización.

Implementación de A: comunicación IoT y acciones del agente

Las acciones posibles A del agente se ejecutan una vez construida la entidad NGSI. El conjunto A comprende: (1) transmitir el mensaje al broker FIWARE mediante HTTP POST o MQTT PUBLISH según el protocolo configurado, (2) registrar el ciclo como exitoso, o (3) registrar el fallo correspondiente. El firmware implementa control explícito del proceso de envío configuración de interfaz WiFi/BLE, establecimiento de conexión, transmisión y verificación de respuesta HTTP 200 o ACK MQTT. Ante un fallo de comunicación, el agente registra el evento

y continúa con el siguiente ciclo sin detenerse, manteniendo operación continua incluso con conectividad intermitente.

Control temporal y confiabilidad operativa

El control temporal y la confiabilidad operativa no son módulos independientes del agente, sino la forma en que la restricción formal Ttotal ≤ Tdeadline y la métrica R definidas en la sección 4.3 se convierten en e mecanismos ejecutables dentro del firmware. Sin este componente, el agente no podría garantizar que actúa dentro de los límites temporales ni que sus ciclos sean trazables; es, por tanto, la pieza que cierra el ciclo de operacionalización del agente sobre Arduino.

La restricción Ttotal ≤ Tdeadline se vuelve operacional mediante la instrumentación del ciclo con millis() en Arduino. Si Tcaptura + Tprocesamiento + Tcomunicación supera Tdeadline, el ciclo se marca como fallo temporal. Para la confiabilidad, el agente mantiene contadores internos: número de ciclos totales, ciclos exitosos y fallos por tipo (percepción, inferencia, estandarización, comunicación, tiempo excedido), que permiten calcular R = Nciclos_exitosos / Nciclos_totales IC_{95}% al final de cada sesión, verificando empíricamente la robustez del agente en condiciones reales.

Desde el punto de vista de adquisición de datos, la integración comienza con la implementación del módulo de percepción sobre Arduino. Cada sensor se gestiona mediante rutinas de lectura específicas, dependiendo si se trata de entradas analógicas o digitales. El agente organiza las lecturas en un vector de características xt, el cual se construye en memoria como una estructura compacta (por ejemplo, un arreglo de valores numéricos). En esta etapa, el agente realiza validaciones básicas orientadas a robustez: detección de lecturas nulas, valores fuera de rango físico y fallos de comunicación con el sensor (cuando el sensor usa un bus o protocolo específico). Esta validación tiene un propósito técnico fundamental: evitar que datos erróneos lleguen al modelo embebido, lo que podría producir predicciones inconsistentes o incluso errores de ejecución si se desbordan rangos numéricos o buffers.

Una vez construido el vector xt, el agente lo prepara para el modelo de aprendizaje automático embebido. Esta preparación incluye la normalización o re-escalamiento requerido por el modelo (cuando aplica), la conversión al tipo numérico adecuado y la carga en el buffer de entrada que utiliza el motor de inferencia. En el entorno TinyML, el modelo se integra como un conjunto de parámetros y operaciones embebidas dentro del firmware. Esto implica que el modelo no se descarga ni se evalúa dinámicamente como en un entorno de servidor, sino que queda enlazado al programa como parte del binario ejecutable. En consecuencia, la ejecución del modelo se implementa como una llamada directa a la rutina de inferencia, la cual recibe el vector xt y retorna la predicción $y\hat{}t$.
Durante la inferencia, el modelo realiza la transformación interna correspondiente a su arquitectura (por ejemplo, operaciones de decisión en árboles o propagación hacia adelante en modelos neuronales compactos), produciendo una salida en un formato definido. En

clasificación, esta salida puede corresponder a puntajes o probabilidades por clase; en regresión, a un valor continuo.

El agente encapsula esta salida en una variable interna y realiza una verificación mínima de consistencia (por ejemplo, confirmar que la clase resultante pertenece al conjunto permitido o que el valor numérico se encuentra en un dominio esperado). Esta verificación no se plantea como una evaluación estadística del modelo, sino como un mecanismo de seguridad embebida para asegurar estabilidad operativa del firmware.

Posteriormente, la integración con FIWARE se materializa a través del módulo de estandarización NGSI. En esta etapa, el agente construye un objeto estructurado que representa una entidad IoT, con atributos asociados tanto a las variables sensadas como a la predicción generada. Este proceso requiere traducir el estado interno del agente (lecturas y predicción) a una representación interoperable, utilizando identificadores de entidad, tipo de entidad y atributos con estructura NGSI. En términos prácticos, esta estandarización implica construir un JSON conforme al modelo de datos adoptado, agregando metadatos esenciales (por ejemplo, marca temporal) y asegurando consistencia en nombres, tipos de dato y estructura. El valor técnico de esta etapa es que el agente no transmite “datos crudos”, sino información contextual que puede integrarse automáticamente en el Orion Context Broker sin transformaciones adicionales del lado servidor.

Con el mensaje NGSI construido, el agente ejecuta el módulo de comunicación. En un entorno Arduino, la comunicación hacia la plataforma IoT no puede asumirse como confiable; por ello, el firmware implementa un control explícito del proceso de envío, que incluye configuración de interfaz (WiFi/MQTT/HTTP según el caso), establecimiento de conexión, transmisión del mensaje y verificación de éxito. En caso de fallo, el agente registra el evento y continúa con el siguiente ciclo en lugar de detenerse. Este comportamiento es esencial para mantener operación continua en sistemas IoT reales, donde la conectividad puede ser intermitente.

Un aspecto central de la implementación embebida es el control temporal. La restricción

$Ttotal \leq Tdeadline$ definida en la formalización matemática se vuelve operacional en Arduino mediante la instrumentación del tiempo de ciclo: el agente mide el instante de inicio y fin del ciclo e identifica cuando el procesamiento, la estandarización o la comunicación provocan una ejecución fuera del límite permitido. Este mecanismo no pretende optimizar el rendimiento en esta sección, sino garantizar que el agente sea capaz de operar bajo un criterio temporal explícito que posteriormente será utilizado para la evaluación experimental y el cálculo de confiabilidad.

Finalmente, el agente incorpora un mecanismo de registro interno orientado a trazabilidad. En un microcontrolador, el registro no se concibe como un log extenso, sino como contadores y marcas mínimas que permiten reconstruir el comportamiento global del agente: número de ciclos, número de ciclos exitosos y conteo de fallos por tipo (lectura, ejecución del modelo,

estandarización, comunicación y tiempo excedido). Este diseño es consistente con el objetivo de demostrar que el agente funciona de manera autónoma y controlada dentro de una infraestructura restringida.

En conjunto, la integración del agente inteligente en Arduino demuestra que el comportamiento definido en este capítulo percepción, ejecución de modelo embebido, estandarización NGSI y comunicación IoT puede ejecutarse de forma completa dentro del microcontrolador, bajo un enfoque modular y determinista. Esta implementación es un paso fundamental para que, en el capítulo de evaluación, sea posible medir la confiabilidad operacional del agente y analizar su desempeño bajo escenarios reales de operación. El Capítulo 5 describe la plataforma web desarrollada para automatizar la generación de este agente, eliminando la necesidad de programación manual en cada etapa del proceso.

Capítulo 5 – Plataforma para la Generación de Agentes Inteligentes

5.1 Introducción

Este capítulo describe la plataforma desarrollada para automatizar la generación de agentes inteligentes embebidos en dispositivos Arduino. La plataforma es el componente que cierra la brecha identificada en el Capítulo 3: integra en un solo flujo el entrenamiento de modelos TinyML, su compresión para hardware embebido, la configuración de la comunicación IoT y la generación automática del firmware, eliminando la necesidad de programación manual en cada etapa. A diferencia de los trabajos revisados en el estado del arte que abordan la inferencia embebida o la integración IoT de forma aislada y requieren intervención técnica especializada en cada etapa, esta plataforma unifica ambas dimensiones en un flujo continuo y accesible.

Una contribución diferencial es la etapa de optimización del modelo: mientras los trabajos comparables del estado del arte aplican cuantización post-entrenamiento de forma manual y separada del despliegue, esta plataforma automatiza la cuantización int8 y la poda estructural del modelo dentro del mismo flujo que genera el firmware, garantizando que el modelo comprimido sea compatible con el hardware objetivo antes de que el usuario descargue el agente.

El capítulo se organiza en cuatro secciones. Primero se describe la arquitectura general del sistema y la relación entre la plataforma y el agente embebido. Segundo se detalla el stack tecnológico y las decisiones de diseño que justifican cada elección. Tercero se describe cada uno de los cinco módulos de la plataforma con su entrada, proceso y salida. Cuarto se explica el flujo completo de generación del agente desde la perspectiva del usuario.

El agente que esta plataforma genera es exactamente el definido en el Capítulo 4: un firmware con cuatro módulos funcionales (percepción, inferencia, estandarización y comunicación) que implementa la formalización matemática descrita. El Capítulo 6 evalúa experimentalmente los agentes producidos por esta plataforma en escenarios reales.

5.2 Arquitectura General del Sistema

El sistema se compone de dos elementos principales que operan en capas distintas: la plataforma web que ejecuta las tareas computacionalmente intensivas y el agente inteligente embebido que ejecuta únicamente las funciones de tiempo real en el microcontrolador. La separación de responsabilidades entre ambas capas es una decisión de diseño fundamental que permite optimizar el uso de recursos en cada extremo (ver figura 4).

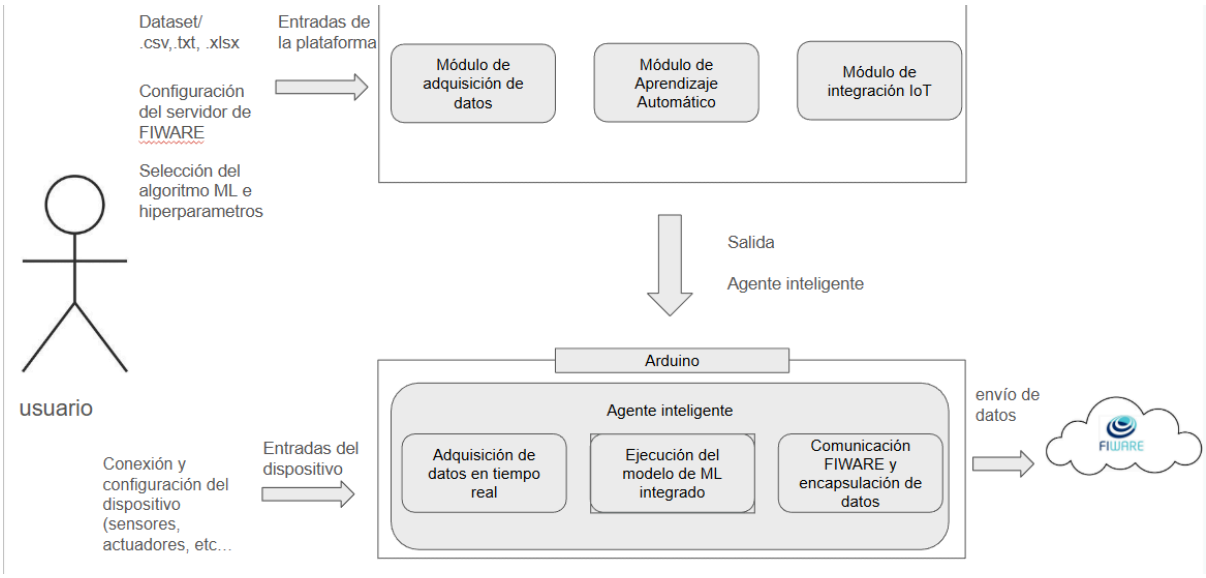

Figura 4. Arquitectura completa de la solución propuesta

La plataforma web recibe datos históricos del usuario, entrena y optimiza el modelo de aprendizaje automático en el servidor, comprime el modelo al formato TFLite cuantizado y genera el código del agente mediante sustitución de parámetros en plantillas Arduino. Todo este procesamiento ocurre una sola vez, antes del despliegue. El resultado es un archivo .ino que el usuario descarga y carga en su dispositivo Arduino mediante el IDE estándar.

El agente embebido, una vez desplegado, opera de forma completamente autónoma: no requiere conexión permanente a la plataforma ni acceso a internet para ejecutar la inferencia. La conectividad solo se usa para transmitir las predicciones al broker FIWARE, y el agente está diseñado para tolerar interrupciones en esa comunicación sin detenerse.

Justificación de la separación de capas: entrenar un modelo Random Forest con 700,000 instancias y validación cruzada k=10 requiere segundos en un servidor moderno, pero sería inviable en un Arduino. A la inversa, ejecutar inferencia en 35 ms con un modelo de 58 KB es trivial en el Nano 33 BLE pero innecesario en un servidor. La plataforma asigna cada tarea al entorno donde puede ejecutarse de forma óptima.

5.3 Stack Tecnológico y Decisiones de Diseño

La plataforma fue desarrollada en R utilizando el framework Shiny para la interfaz web. Esta elección responde a tres criterios: (1) R cuenta con el ecosistema más completo de librerías

estadísticas y de aprendizaje automático para el perfil de usuario objetivo investigadores y científicos de datos, (2) Shiny permite construir interfaces web reactivas sin requerir conocimientos de JavaScript o frameworks frontend separados, y (3) el despliegue en shinyapps.io no requiere infraestructura de servidor propia, lo que reduce significativamente la barrera de adopción.

La única excepción al stack R es el módulo de compresión TinyML, que invoca un script Python mediante la función system() de R. Esta decisión es necesaria porque TensorFlow Lite Converter la herramienta estándar para cuantización int8 post-entrenamiento no tiene implementación nativa en R. La llamada es transparente para el usuario: la plataforma gestiona la invocación del script y recupera el modelo comprimido sin exponer esta dependencia en la interfaz. La Tabla 3 resume el stack tecnológico completo por capa, indicando la tecnología, las librerías específicas y la función de cada componente dentro de la plataforma:

Capa	Tecnología	Librería / Herramienta	Función en la plataforma
Interfaz de usuario	R + Shiny	shiny, shinydashboard	Interfaz web reactiva: carga de datos, configuración de modelos, visualización de resultados y descarga del agente.
Visualización	R + Shiny	ggplot2, plotly, DT	Gráficas interactivas del dataset, curvas de aprendizaje, matrices de confusión y visualización del árbol de decisión.
Motor ML	R	caret, randomForest, rpart, keras (R)	Entrenamiento de RF, DT y Red Neuronal con validación cruzada k=10 y búsqueda en cuadrícula de hiperparámetros.
Compresión TinyML	Python (llamado desde R)	TFLite Converter, TensorFlow	Cuantización post-entrenamiento int8 (PTQ) y poda estructural del modelo. Invocado mediante system() desde R.
Generación de código	R	Plantillas .ino + glue / paste()	Sustitución de parámetros en plantillas Arduino: modelo embebido, lógica de inferencia y módulo de comunicación NGSI v2.

Despliegue	Shiny local / shinyapps.io	rsconnect, shinyapps.io	Ejecución local durante el desarrollo. Publicación en shinyapps.io para acceso remoto sin infraestructura adicional.

Tabla 3. Stack tecnológico de la plataforma por capa funcional.

Sobre la elección de R frente a Python/Flask: una plataforma equivalente en Python requeriría un frontend separado (HTML/CSS/JS o React), un backend Flask/FastAPI, y gestión de sesiones y estado entre ambos. Shiny integra todo en un solo entorno reactivo en R, reduciendo la complejidad de desarrollo y mantenimiento. Para el perfil de usuario de esta plataforma investigadores con experiencia en análisis de datos R es además el entorno más familiar.

5.4 Módulos de la Plataforma

La plataforma se organiza en cinco módulos secuenciales. Cada módulo tiene una entrada bien definida, un proceso específico y una salida que alimenta al módulo siguiente. La Tabla 4 presenta la estructura de los cinco módulos antes de describirlos en detalle:

Módulo	Entrada	Proceso	Salida
M1 — Carga y visualización	Archivo CSV con variables del sensor y etiquetas	Validación de formato, estadísticos descriptivos, gráficas ggplot2/plotly	Dataset validado y explorado, listo para entrenamiento
M2 — Entrenamiento ML	Dataset validado + selección de algoritmo e hiperparámetros	caret con train(); k=10 cross-validation; Grid Search automático	Modelo entrenado con métricas F1/R^2/RMSE; curvas de aprendizaje
M3 — Compresión TinyML	Modelo entrenado en R + placa Arduino seleccionada	Script Python: TFLite Converter + PTQ int8; verificación RAM/Flash por placa	Modelo comprimido (.tflite) con reporte de compatibilidad hardware
M4 — Configuración IoT	Parámetros FIWARE: URL broker, ID entidad, atributos NGSI	Formulario Shiny; validación de parámetros; generación de configuración embebible	Configuración IoT lista para integrar en el sketch .ino

M5 — Generación del agente	Modelo comprimido + configuración IoT + plantilla .ino	Sustitución de parámetros en plantilla; empaquetado del sketch completo	Archivo sketch_agente.ino descargable, listo para compilar y cargar en Arduino

Tabla 4. Módulos de la plataforma: entrada, proceso y salida de cada etapa.

5.4.1 Módulo 1: Carga y Visualización de Datos

El usuario carga un archivo CSV con los datos históricos del sensor. La plataforma valida automáticamente el formato (tipos de columna, valores faltantes, rango de variables) y rechaza datasets que no cumplan los requisitos mínimos de estructura. Una vez validado, el módulo presenta tres vistas de exploración:

- Tabla interactiva (DT): muestra las primeras filas del dataset con filtros y ordenamiento, permitiendo al usuario verificar que los datos se cargaron correctamente.
- Distribución de variables (ggplot2): histogramas y boxplots de cada variable del sensor, para identificar outliers y distribuciones sesgadas antes del entrenamiento.
- Evolución temporal (plotly): gráfica interactiva de las variables en el tiempo, con zoom y selección de rangos, útil para identificar patrones estacionales o eventos anómalos.

Esta etapa de exploración es importante porque permite al usuario tomar decisiones informadas sobre el preprocesamiento antes de entrenar: si una variable tiene muchos outliers o distribución muy sesgada, el usuario puede optar por normalizarla o filtrarla antes de continuar. La figura 6 muestra la interfaz correspondiente a este módulo.

Figura 6. Interfaz de la plataforma web para carga y visualización de datos históricos de sensores.

5.4.2 Módulo 2: Entrenamiento ML y Validación del Modelo

El usuario selecciona el algoritmo de aprendizaje automático y configura sus hiperparámetros desde la interfaz. La plataforma soporta tres tipos de modelos, cada uno con sus hiperparámetros configurables:

- Random Forest (randomForest): n_estimators (número de árboles), max_depth (profundidad máxima), max_features (características por split) y class_weight para balanceo de clases.
- Árbol de Decisión (rpart): max_depth, criterion (Gini o entropía) y ccp_alpha para poda controlada post-entrenamiento.
- Red Neuronal (keras en R): número de capas ocultas, neuronas por capa, función de activación, tasa de aprendizaje, dropout y número de épocas.

El entrenamiento se ejecuta mediante el paquete caret, que unifica la interfaz de entrenamiento para los tres tipos de modelos y gestiona automáticamente la validación cruzada k=10 y la búsqueda en cuadrícula de hiperparámetros. Una vez finalizado, el módulo presenta:

- Métricas de evaluación: precisión, sensibilidad, especificidad, exactitud y F1 para clasificación; R^2 y RMSE para regresión, calculados sobre el conjunto de validación con k=10.
- Curvas de aprendizaje (ggplot2): evolución del error de entrenamiento y validación por época (red neuronal) o por número de árboles (Random Forest).
- Visualización del modelo (plotly/rpart.plot): árbol de decisión graficado con nodos y reglas de decisión; importancia de variables para Random Forest.

La figura 7 presenta la interfaz del módulo de entrenamiento con los controles de selección de modelo, los paneles de métricas y las visualizaciones generadas.

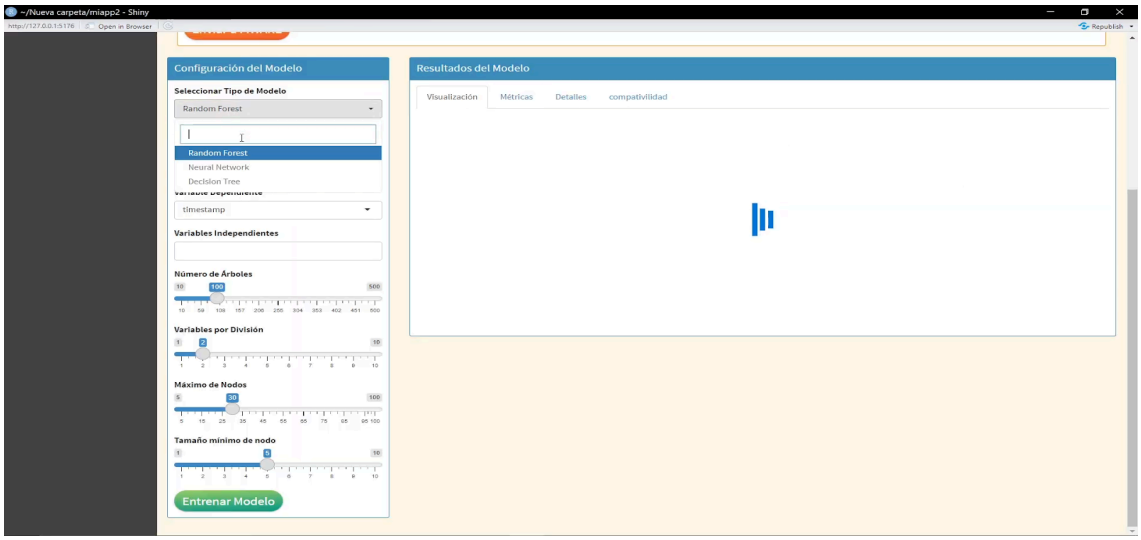

Figura 7 Interfaz de la plataforma web para selección de modelo ML e hiperparámetros configurables, con métricas de evaluación y curvas de aprendizaje.

5.4.3 Módulo 3: Compresión TinyML y Verificación de Compatibilidad

Una vez seleccionado el modelo con mejor desempeño, la plataforma ejecuta el proceso de compresión para adaptarlo a las restricciones del hardware embebido. Este módulo tiene dos etapas:

Primero, la compresión del modelo. La plataforma invoca un script Python mediante system() de R, que utiliza TensorFlow Lite Converter para aplicar cuantización post-entrenamiento a enteros de 8 bits (PTQ int8). Este proceso convierte los pesos del modelo de punto flotante de 32 bits a enteros de 8 bits, reduciendo el tamaño del modelo en un factor aproximado de 4×. Para el Árbol de Decisión y el Random Forest, la conversión serializa la estructura del árbol como un arreglo de bytes compatibles con TFLite Micro. El modelo resultante se almacena como archivo .tflite.

Segundo, la verificación de compatibilidad. La plataforma calcula el tamaño del modelo comprimido y el tensor arena necesario para la inferencia, y los compara con la RAM y Flash disponibles en la placa seleccionada. Si el modelo excede los límites de la placa, la plataforma rechaza la configuración y sugiere un modelo más ligero o una placa con mayor capacidad. Este mecanismo garantiza que solo se generen agentes ejecutables en el hardware de destino.

El reporte de compatibilidad muestra al usuario una tabla con: tamaño del modelo en KB, RAM estimada durante inferencia, Flash utilizada, margen disponible y un indicador de viabilidad (verde / amarillo / rojo) para cada placa disponible. La figura 8 muestra este reporte de compatibilidad generado por la plataforma.

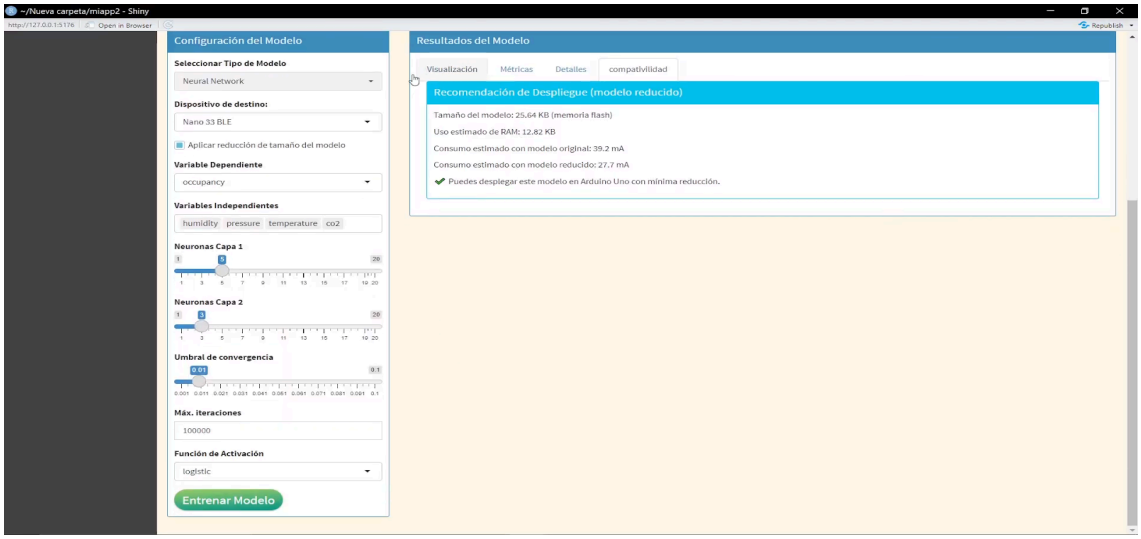

Figura 8. Interfaz de la plataforma web con el reporte de compatibilidad entre el modelo comprimido y las placas Arduino disponibles.

5.4.4 Módulo 4: Configuración de la Comunicación IoT

El usuario define los parámetros necesarios para la integración con FIWARE mediante un formulario en la interfaz Shiny:

- URL del broker: dirección del Orion Context Broker (por ejemplo, http://192.168.1.100:1026).
- ID de entidad: identificador único del dispositivo en formato NGSI (urn:ngsi-ld:Device:001).
- Tipo de entidad: categoría del dispositivo en el modelo de datos (IoTDevice, WeatherStation, etc.).

La plataforma valida que los parámetros tengan el formato correcto antes de continuar y genera una vista previa del mensaje NGSI v2 que el agente enviará en cada ciclo, permitiendo al usuario verificar la estructura antes de la generación del firmware. La figura 9 muestra el formulario de configuración IoT disponible en la plataforma.

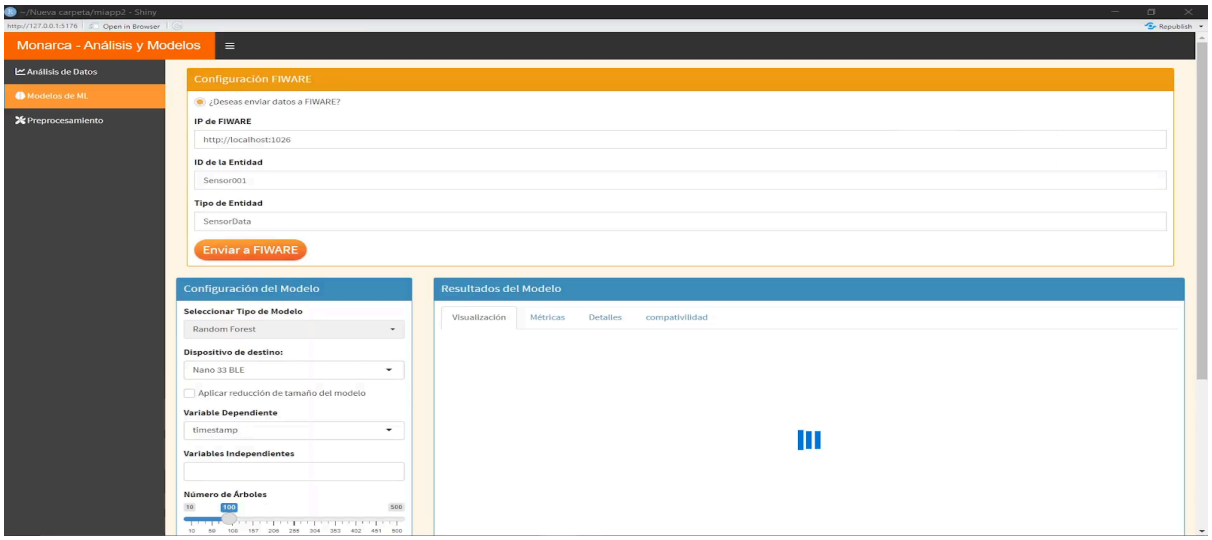

Figura 9 Interfaz de la plataforma web para la configuración de la comunicación IoT con FIWARE.

5.4.5 Módulo 5: Generación Automática del Agente

Con el modelo comprimido y la configuración IoT completos, la plataforma genera el firmware del agente mediante sustitución de parámetros en plantillas Arduino (.ino). Las plantillas están estructuradas en tres secciones que corresponden a los módulos del agente definidos en el Capítulo 4:

- Sección de inferencia: incluye el arreglo de bytes del modelo TFLite, la inicialización del tensor arena y la llamada al intérprete TFLite Micro con el vector de entrada x□.

- Sección de estandarización: construye el mensaje JSON NGSI v2 con los atributos configurados en el Módulo 4, sustituyendo los valores de x□ y ŷ□ en tiempo de ejecución.
- Sección de comunicación: implementa la transmisión HTTP POST o MQTT PUBLISH con los parámetros del broker configurados, incluyendo manejo de errores y registro de fallos.

La sustitución de parámetros se realiza en R mediante las funciones glue() y paste(), que reemplazan los marcadores de posición de la plantilla con los valores específicos del modelo entrenado, la placa seleccionada y la configuración IoT. El resultado es un archivo sketch_agente.ino autocontenido que el usuario descarga con un clic y carga directamente en su Arduino mediante el IDE estándar o Arduino CLI, sin modificaciones manuales adicionales. La figura 10 muestra la pantalla de descarga del agente generado con el resumen de la configuración embebida.

El sketch generado implementa exactamente el ciclo operativo de seis etapas descrito en la sección 4.4: inicio del ciclo, percepción, inferencia, estandarización, control temporal y comunicación.

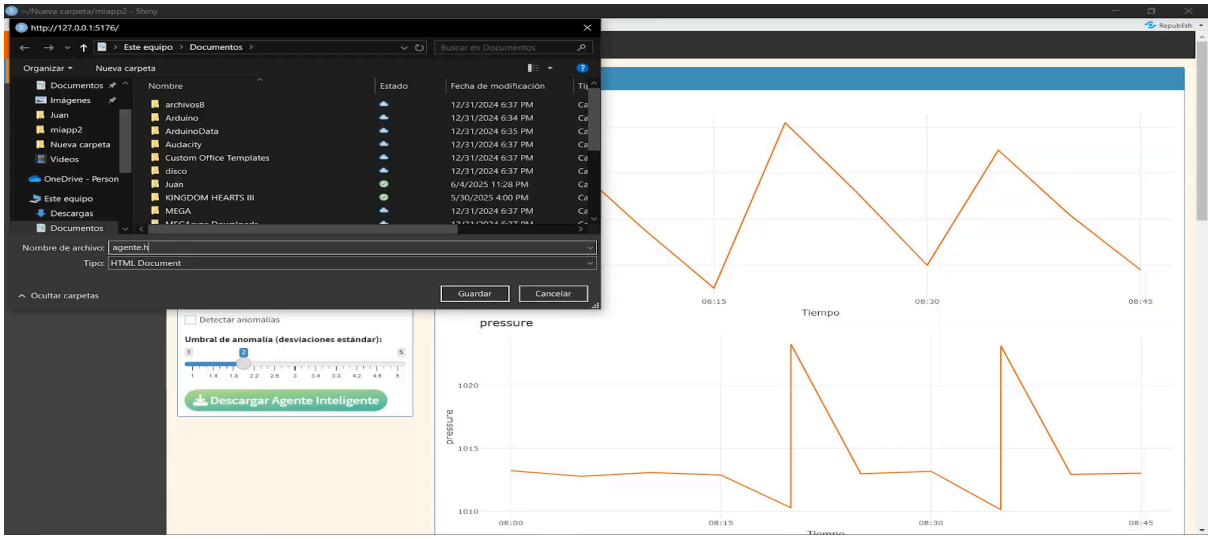

Figura 10. Interfaz de la plataforma web con el agente generado listo para descargar.

5.5 Flujo Completo de Generación del Agente

Desde la perspectiva del usuario, el flujo completo de generación del agente se resume en cinco pasos secuenciales que la interfaz Shiny guía de forma explícita:

1. Cargar dataset — el usuario sube el archivo CSV con los datos históricos del sensor. La plataforma valida el formato, muestra estadísticos descriptivos y gráficas de exploración, y confirma que el dataset es apto para el entrenamiento.

2. Seleccionar y entrenar modelo — el usuario elige el algoritmo (RF, DT o NN), ajusta los hiperparámetros desde la interfaz y lanza el entrenamiento. La plataforma ejecuta caret con k=10 cross-validation, muestra las métricas y las visualizaciones del modelo, y el usuario selecciona la configuración con mejor desempeño.
3. Verificar compatibilidad de hardware — el usuario selecciona la placa Arduino de destino. La plataforma comprime el modelo con TFLite Converter, calcula el consumo de RAM y Flash, y muestra el reporte de compatibilidad. Si la placa no es compatible, el usuario puede elegir un modelo más ligero o una placa con mayor capacidad.
4. Configurar comunicación IoT — el usuario completa el formulario con los parámetros FIWARE (broker, entidad, atributos, protocolo) y revisa la vista previa del mensaje NGSI v2 que el agente generará en cada ciclo.
5. Descargar el agente — la plataforma genera el sketch .ino con todos los componentes integrados (modelo embebido, inferencia, NGSI v2, comunicación IoT) y lo pone a disposición para descarga. El usuario lo carga en el Arduino mediante el IDE estándar.

Tiempo total del proceso: en las pruebas realizadas, el flujo completo desde la carga del dataset hasta la descarga del agente tomó entre 3 y 8 minutos, dependiendo del tamaño del dataset y el tipo de modelo seleccionado. El entrenamiento con Random Forest y k=10 folds sobre 700,000 instancias tomó aproximadamente 4 minutos en hardware de escritorio estándar. La compresión TFLite adicional menos de 30 segundos.

Este capítulo describió la plataforma web desarrollada en R/Shiny para la generación automática de agentes inteligentes embebidos. La plataforma integra cinco módulos secuenciales carga de datos, entrenamiento ML, compresión TinyML, configuración IoT y generación del firmware en un flujo guiado que elimina la necesidad de programación manual en cualquiera de sus etapas.

Las decisiones de diseño más relevantes son tres: la elección de R/Shiny como entorno único de desarrollo e interfaz (frente a arquitecturas cliente-servidor separadas), la delegación de la compresión TinyML a un script Python invocado desde R (aprovechando TFLite Converter sin sacrificar la integración del resto del flujo), y el uso de plantillas .ino con sustitución de parámetros (que garantiza que el firmware generado implementa exactamente la arquitectura del agente definida en el Capítulo 4).

El Capítulo 6 evalúa experimentalmente los agentes generados por esta plataforma en dos casos de estudio reales, midiendo su desempeño predictivo, eficiencia operativa y confiabilidad en cuatro plataformas de hardware Arduino distintas.

Capítulo 6 – Pruebas y resultados

6.1 Introducción

Este capítulo presenta el proceso experimental completo que se llevó a cabo para validar el funcionamiento del agente inteligente propuesto. Las pruebas están organizadas en tres fases progresivas: la primera verifica que la plataforma genera agentes funcionales y los despliega correctamente en hardware real; la segunda evalúa el desempeño de dichos agentes en dos escenarios de aplicación real con relevancia práctica; y la tercera mide el rendimiento del agente en cuatro placas Arduino con características de hardware distintas, utilizando el mismo modelo de referencia para garantizar una comparación justa.

El hilo conductor de las tres fases es siempre el mismo agente inteligente: un componente embebido capaz de percibir datos de sensores, ejecutar inferencia mediante un modelo de aprendizaje automático ligero (TinyML), estandarizar los resultados en formato NGSI v2 y transmitirlos a la plataforma FIWARE. Cada fase añade una capa de evidencia sobre una dimensión distinta: corrección funcional, utilidad en escenarios reales y portabilidad entre hardware heterogéneo.

La Tabla 5 resume las pruebas realizadas, el hardware involucrado, los conjuntos de datos y las métricas de evaluación utilizadas en cada fase, para facilitar la lectura del capítulo.

Fase	Objetivo	Hardware / Dataset	Modelo evaluado	Métricas clave
Fase 1 – Instalación y configuración	Verificar que la plataforma genera y despliega agentes correctamente	Nano 33 BLE / Dataset de ocupación de habitación	Árbol de Decisión	Despliegue exitoso, latencia, transmisión FIWARE
Fase 2 – Caso de estudio 1: Detección de caídas	Evaluar el agente en un escenario de seguridad humana en tiempo real	Nano 33 BLE + IMU LSM9DS1 / 700,000 instancias	Umbrales, RF, NN, DT	Precisión, sensibilidad, especificidad, exactitud, F1

Fase	Objetivo	Hardware / Dataset	Modelo evaluado	Métricas clave
Fase 2 – Caso de estudio 2: Evapotranspiración	Evaluar el agente en un escenario de monitoreo ambiental agrícola	Nano 33 BLE + DHT22 + anemómetro / 1,000,000 instancias	PG, RF, NN, DT	R^2, RMSE, tamaño del modelo
Fase 3 – Rendimiento y compatibilidad	Medir el impacto del hardware sobre latencia, memoria y tasa de transmisión	Uno, Nano 33 BLE, ESP32, Portenta H7 / ambos datasets	Random Forest (modelo de referencia fijo)	Latencia, tasa (reg/s), RAM %, Flash %

Tabla 5. Resumen del plan experimental por fases.

6.2 Fase 1: Generación y Despliegue del Agente Inteligente

El objetivo de esta fase no es evaluar el desempeño predictivo del agente, sino verificar que la plataforma web genera correctamente todos los componentes del agente modelo TinyML, lógica de inferencia y módulo de comunicación NGSI v2 y que el código resultante se instala y ejecuta de forma estable en hardware real. Para ello se utilizó un conjunto de datos de ocupación de habitaciones, con variables de temperatura, humedad, presión y nivel de CO_2, cuya tarea de clasificación (habitación ocupada o vacía) es suficientemente sencilla para centrar la atención en la corrección del proceso de generación y no en la dificultad del problema.

6.2.1 Configuración experimental

Se cargó el conjunto de datos de ocupación en la plataforma web, que contiene las siguientes variables de entrada:

- timestamp: marca de tiempo de la medición.
- humidity: humedad relativa (%).
- pressure: presión atmosférica (hPa).
- temperature: temperatura ambiente (°C).

- CO_2 Level: concentración de dióxido de carbono (ppm).
- occupancy: variable objetivo — 1 (habitación ocupada), 0 (habitación vacía).

Desde la interfaz de la plataforma se entrenó un modelo de Árbol de Decisión con validación cruzada k=10 y ajuste de hiperparámetros mediante búsqueda en cuadrícula. Una vez entrenado y evaluado, la plataforma generó automáticamente el código del agente, que integra los tres módulos funcionales:

1. Módulo de adquisición: lectura de los sensores conectados al Arduino Nano 33 BLE (temperatura, humedad, presión y CO_2) a través de los interfaces I^2C y analógico.
2. Módulo de inferencia: ejecución local del modelo TinyML mediante el intérprete TensorFlow Lite Micro, con el modelo cuantizado a enteros de 8 bits (int8) e incrustado como arreglo de bytes en el firmware.
3. Módulo de comunicación: encapsulación de los datos del sensor y la predicción del modelo en una entidad NGSI v2, y transmisión al Orion Context Broker de FIWARE mediante el protocolo MQTT.

El agente generado se compiló y cargó en un Arduino Nano 33 BLE. La plataforma FIWARE se configuró con el Orion Context Broker, CrateDB como base de datos de series temporales, y Grafana como herramienta de visualización.

6.2.2 Resultados de la Fase 1

El agente se ejecutó de forma continua durante la sesión de prueba sin presentar errores de compilación ni fallos en tiempo de ejecución. Se verificaron los tres aspectos fundamentales del ciclo del agente:

- Adquisición correcta: el dispositivo leyó satisfactoriamente los sensores integrados y generó el vector de características x□ en cada ciclo de inferencia.
- Inferencia local: el modelo TinyML ejecutó la inferencia con latencias inferiores a 100 ms, confirmando que el código generado por la plataforma es compatible con los recursos del hardware seleccionado.
- Integración extremo a extremo: las predicciones fueron encapsuladas en entidades NGSI v2 y transmitidas correctamente al Orion Context Broker de FIWARE. Los registros se almacenaron en CrateDB y se visualizaron en tiempo real en el dashboard de Grafana, validando el flujo completo de datos.

Esta fase confirma que la plataforma cumple su función principal: automatizar la generación de un agente inteligente funcional sin que el usuario deba escribir manualmente el código de inferencia, la serialización NGSI ni la lógica de comunicación. A partir de esta validación, las fases siguientes evalúan el agente en escenarios de mayor exigencia y complejidad.

6.3 Fase 2: Evaluación en Escenarios de Aplicación Real

La segunda fase experimental tuvo como propósito validar el funcionamiento del agente inteligente en escenarios de aplicación real. Para ello, se seleccionaron dos casos de estudio con naturalezas distintas: la detección de caídas en adultos mayores y la estimación de evapotranspiración de referencia (ETo) en un contexto agrícola.

Estos escenarios fueron seleccionados porque representan problemas donde la autonomía del dispositivo, la inferencia local y la integración con plataformas IoT pueden aportar valor práctico. El primer caso corresponde a una tarea de clasificación binaria, en la que el agente debe identificar si un patrón de movimiento corresponde a una actividad normal o a una caída. El segundo caso corresponde a una tarea de regresión, en la que el agente debe estimar un valor continuo de evapotranspiración a partir de variables ambientales.

Para mejorar la claridad metodológica, ambos casos de estudio se presentan mediante un protocolo experimental por fases. Esta estructura permite describir de forma ordenada la configuración del dispositivo, la carga del dataset, la evaluación de modelos, la selección del algoritmo, el despliegue del agente y su ejecución en tiempo real.

En ambos casos, los modelos fueron entrenados y evaluados dentro de la plataforma web, posteriormente optimizados para su despliegue embebido y finalmente integrados en un agente inteligente ejecutable sobre Arduino. De esta manera, se valida el flujo completo propuesto en la tesis: adquisición de datos, entrenamiento del modelo, generación automática del agente, inferencia local y comunicación interoperable con FIWARE.

6.3.1 Caso de Estudio 1: Detección de Caídas en Adultos Mayores

La detección oportuna de caídas en adultos mayores representa un problema relevante en el área de salud y asistencia remota. Una caída no detectada a tiempo puede derivar en lesiones graves o complicaciones médicas, por lo que resulta importante contar con dispositivos capaces de identificar este tipo de eventos de manera local y en tiempo real.

El objetivo de este caso de estudio fue evaluar si un agente inteligente embebido en un dispositivo Arduino puede detectar caídas a partir de señales de aceleración, utilizando modelos de aprendizaje automático ejecutados localmente y comunicando los resultados hacia una plataforma IoT.

6.3.1.1 Protocolo experimental por fases

El experimento de detección de caídas se organizó en seis fases secuenciales. Esta organización permite mostrar de manera clara el proceso seguido desde la configuración del dispositivo hasta la ejecución del agente inteligente en tiempo real.

Fase 1. Configuración del dispositivo

En esta primera fase se configuró el hardware utilizado para la adquisición de datos y la ejecución del agente inteligente. Se empleó una placa Arduino Nano 33 BLE, equipada con la

unidad inercial LSM9DS1, la cual integra acelerómetro, giroscopio y magnetómetro en un solo chip comunicado mediante el protocolo I²C.

Para este caso de estudio se utilizaron principalmente las lecturas del acelerómetro en los tres ejes espaciales: x, y, z. El dispositivo fue configurado para registrar señales de aceleración con una frecuencia de muestreo de 55 Hz, equivalente a una lectura aproximadamente cada 200 ms.

Esta frecuencia permite capturar variaciones rápidas en el movimiento corporal, lo cual resulta necesario para diferenciar una actividad cotidiana de un evento de caída. El uso de la placa Arduino Nano 33 BLE permitió ejecutar el agente directamente en el dispositivo, manteniendo un equilibrio entre capacidad de procesamiento, memoria disponible y portabilidad.

Fase 2. Carga y preparación del dataset

En la segunda fase se cargó en la plataforma el conjunto de datos correspondiente al caso de estudio de detección de caídas. El dataset contiene 700,000 instancias, cada una compuesta por una marca de tiempo, las aceleraciones en los ejes x, y, z, y una etiqueta binaria que indica si el registro corresponde a una actividad normal o a una caída.

La estructura del conjunto de datos se presenta en la Tabla 6. La variable etiqueta se definió como una salida binaria, donde el valor 0 representa una actividad normal y el valor 1 representa un evento de caída.

timestamp	aceleración_x (g)	aceleración_y (g)	aceleración_z (g)	etiqueta (0/1)
2025-05-01 10:00:00	0.12	-0.98	9.81	0
2025-05-01 10:00:01	1.45	-1.20	2.35	1
2025-05-01 10:00:02	0.03	-0.02	9.76	0
2025-05-01 10:00:03	1.67	-0.80	3.12	1

Tabla 6. Muestra representativa del dataset de detección de caídas.

Antes del entrenamiento, las señales fueron procesadas mediante ventanas deslizantes de 2 segundos, con un solapamiento del 50 %. Este procedimiento permitió transformar las lecturas individuales en segmentos temporales representativos del movimiento.

De cada ventana se extrajeron características estadísticas, tales como media, desviación estándar, varianza por eje y magnitud del vector de aceleración. Este procesamiento permitió que los modelos analizaran patrones de movimiento y no únicamente valores aislados de aceleración.

Fase 3. Evaluación de modelos en la plataforma

En la tercera fase, el dataset preparado fue utilizado dentro de la plataforma para entrenar y evaluar diferentes estrategias de decisión. Se comparó un método clásico basado en umbrales con tres modelos de aprendizaje automático: Random Forest, Red Neuronal y Árbol de Decisión.

La evaluación se realizó mediante validación cruzada con k=10 particiones, con el objetivo de obtener una estimación más robusta del desempeño de cada modelo. Asimismo, se aplicó búsqueda en cuadrícula para ajustar los hiperparámetros principales de cada algoritmo.

En el caso de Random Forest, se evaluaron diferentes configuraciones de número de árboles y profundidad máxima. Para la Red Neuronal, se probaron arquitecturas compactas con una o dos capas ocultas, considerando funciones de activación como ReLU y tanh. Para el Árbol de Decisión, se ajustó la profundidad máxima y el criterio de división.

Los modelos de aprendizaje automático fueron posteriormente optimizados para su despliegue embebido mediante técnicas de reducción de tamaño. Entre estas técnicas se consideraron la cuantización post-entrenamiento a enteros de 8 bits y la poda estructural, con el propósito de reducir el tamaño del modelo y garantizar su compatibilidad con la memoria disponible en el dispositivo Arduino.

Fase 4. Selección del mejor algoritmo

En la cuarta fase se compararon los resultados obtenidos por los modelos evaluados. Para ello, se utilizaron métricas de clasificación como precisión, sensibilidad, especificidad, exactitud y F1-score.

Aunque el método de umbrales presentó un desempeño alto como línea base, el modelo Random Forest reducido mostró el mejor comportamiento entre los modelos de aprendizaje automático. Este modelo mantuvo un desempeño competitivo y una estructura suficientemente ligera para su despliegue en el Arduino Nano 33 BLE.

La selección de Random Forest resulta relevante porque el objetivo de la tesis no se limita a detectar caídas mediante reglas fijas, sino a demostrar que la plataforma puede generar agentes inteligentes con modelos de IA embebidos y ejecutables en hardware restringido.

Además, Random Forest ofreció un equilibrio adecuado entre desempeño predictivo, tamaño del modelo, estabilidad y viabilidad de implementación. Por esta razón, fue seleccionado como modelo principal para la generación del agente inteligente en este caso de estudio.

Fase 5. Despliegue del agente inteligente en el dispositivo

En la quinta fase, la plataforma generó automáticamente el agente inteligente a partir del modelo seleccionado. Este agente incluyó tres componentes principales: el modelo de inferencia, las funciones de adquisición y procesamiento de datos del sensor, y el módulo de comunicación para estructurar y transmitir los resultados bajo el formato NGSI v2 compatible con FIWARE.

El código generado fue compilado y cargado en el Arduino Nano 33 BLE. Una vez desplegado, el agente quedó configurado para ejecutar el ciclo completo de operación: lectura de datos del acelerómetro, procesamiento de la señal, inferencia local del modelo, generación de la salida predictiva y preparación del resultado para su comunicación hacia la plataforma IoT.

Esta fase permitió comprobar que la plataforma no solo entrena modelos, sino que también genera un componente funcional capaz de operar directamente en el dispositivo. De esta manera, el Arduino deja de comportarse únicamente como un nodo de sensado y se convierte en un agente inteligente con capacidad de análisis local.

Fase 6. Ejecución del agente en tiempo real y resultado

Finalmente, en la sexta fase se ejecutó el agente inteligente en tiempo real. Durante esta etapa, el dispositivo leyó continuamente las señales del acelerómetro, generó el vector de entrada correspondiente, ejecutó la inferencia local y clasificó cada ventana de movimiento como actividad normal o caída.

El resultado de esta fase permitió comprobar que el agente inteligente podía operar de forma autónoma sobre el dispositivo Arduino, sin depender de procesamiento externo para realizar la clasificación. Además, los resultados generados por el agente fueron estructurados como entidades de contexto en formato NGSI v2, permitiendo su integración con FIWARE para almacenamiento, consulta y posterior graficación.

Esta validación confirma el flujo completo propuesto en la tesis: percepción del entorno, inferencia local, decisión embebida y comunicación interoperable con una plataforma IoT.

6.3.1.2 Resultados

Los resultados obtenidos en el caso de estudio de detección de caídas permiten comparar el desempeño del agente inteligente frente al método de umbrales utilizado como línea base. La Tabla 7 presenta las métricas de clasificación obtenidas para cada estrategia evaluada.

Modelo	Tamaño (KB)	Precisión	Sensibilidad	Especificidad	Exactitud
Método de Umbrales	**—**	**94.44 %**	**94.44 %**	**94.44 %**	**94.44 %**
Random Forest (red.)	58	93.75 %	90.00 %	94.00 %	92.00 %
Red Neuronal (red.)	60	90.00 %	88.00 %	91.00 %	89.00 %
Árbol de Decisión (red.)	45	89.00 %	87.50 %	90.00 %	88.20 %

Tabla 7. Resultados de clasificación para el caso de estudio de detección de caídas (validación cruzada k=10).

El método de umbrales alcanzó una precisión, sensibilidad y exactitud del 94.44 % en todas las métricas, lo que refleja que, para este conjunto de datos, los patrones de aceleración durante una caída son suficientemente diferenciados de la actividad normal como para ser detectados por una regla simple de magnitud. Este resultado establece una línea base exigente frente a la cual se comparan los modelos de aprendizaje automático.

Entre los modelos de IA ligera, el Random Forest reducido obtuvo el mejor desempeño, con una precisión del 93.75 %, especificidad del 94.00 % y exactitud del 92.00 %. La ligera reducción respecto al método de umbrales del orden de 1-2 puntos porcentuales se explica por el proceso de compresión (poda y cuantización): el modelo original entrenado en servidor supera al método de umbrales en escenarios con variabilidad de usuario, pero la reducción de complejidad necesaria para su ejecución embebida introduce una pequeña pérdida de precisión. Esta compensación entre tamaño del modelo y rendimiento predictivo es inherente al paradigma TinyML y constituye uno de los hallazgos relevantes de este estudio.

La Red Neuronal reducida y el Árbol de Decisión mostraron un desempeño ligeramente inferior, con exactitudes del 89.00 % y 88.20 % respectivamente. En el caso de la red neuronal, la limitación proviene de la reducción drástica de la arquitectura: una red de dos capas con 16 y 8 neuronas captura relaciones no lineales entre los ejes de aceleración, pero pierde capacidad representativa frente al Random Forest al aplicar cuantización. El Árbol de Decisión, al ser el

modelo más ligero (45 KB), es también el de menor exactitud, pero se posiciona como la única opción viable para placas con restricciones de memoria severas, como el Arduino Uno.

Las predicciones de los tres agentes fueron encapsuladas en entidades NGSI v2 y transmitidas en tiempo real al Orion Context Broker de FIWARE, donde se almacenaron en CrateDB y se visualizaron en Grafana. La integración IoT funcionó de forma estable durante todas las sesiones de prueba, con transmisión de datos continua sin pérdida de paquetes.

6.3.2 Caso de Estudio 2: Estimación de Evapotranspiración (ETo)

La evapotranspiración de referencia (ETo) es una variable relevante para la gestión eficiente del riego agrícola, ya que permite estimar la cantidad de agua que requiere un cultivo bajo determinadas condiciones ambientales. Tradicionalmente, su cálculo requiere el uso de modelos físicos y múltiples variables meteorológicas, lo cual puede resultar complejo para su implementación directa en dispositivos embebidos de bajo costo.

El objetivo de este caso de estudio fue evaluar si un agente inteligente embebido en Arduino puede estimar la ETo a partir de variables ambientales simples medidas localmente, tales como temperatura, humedad relativa y velocidad del viento. A diferencia del caso de detección de caídas, este experimento corresponde a una tarea de regresión, ya que la salida del modelo es un valor continuo expresado en mm/día.

6.3.2.1 Protocolo experimental por fases

El experimento de estimación de evapotranspiración se organizó en seis fases secuenciales. Esta estructura permite describir de manera clara el proceso seguido desde la configuración del dispositivo hasta la ejecución del agente inteligente en condiciones de operación agrícola.

Fase 1. Configuración del dispositivo

En la primera fase se configuró el hardware utilizado para la captura de variables meteorológicas y la ejecución del agente inteligente. Se empleó una placa Arduino Nano 33 BLE, conectada a un sensor DHT22 para la medición de temperatura y humedad relativa, un anemómetro digital para registrar la velocidad del viento y un módulo de comunicación WiFi para la transmisión de datos hacia la plataforma IoT.

La selección de estos sensores responde a la necesidad de estimar la evapotranspiración de referencia a partir de variables ambientales simples y medibles localmente. De esta forma, el dispositivo puede operar como un nodo inteligente de bajo costo, capaz de capturar información del entorno agrícola y procesarla directamente en el borde.

Este enfoque permite reducir la dependencia de sistemas externos de procesamiento, ya que el agente realiza la estimación localmente y solo transmite el resultado estructurado hacia la plataforma IoT.

Fase 2. Carga y preparación del dataset

En la segunda fase se cargó en la plataforma el conjunto de datos correspondiente al caso de estudio de evapotranspiración. El dataset contiene 1,000,000 de instancias con muestreo

diario, integrado por las variables fecha, temperatura, humedad relativa, velocidad del viento y ETo expresada en mm/día.

La estructura del conjunto de datos se presenta en la Tabla 8. La variable de salida fue ETo, por lo que el problema se abordó como una tarea de regresión.

Fecha	Temperatura (°C)	Humedad (%)	Velocidad viento (m/s)	ETo (mm/día)
2025-04-15	28.4	67.0	1.2	4.21
2025-04-16	29.1	62.0	1.5	4.40
2025-04-17	27.3	69.0	0.9	3.95
2025-04-18	30.0	61.0	1.8	4.55

Tabla 8. Muestra representativa del dataset de evapotranspiración (ETo en mm/día).

Antes del entrenamiento, la plataforma permitió validar la estructura del dataset, identificar las variables de entrada, definir la variable objetivo y preparar los datos para su evaluación experimental. Esta etapa fue necesaria para asegurar que los modelos recibieran datos consistentes y compatibles con el proceso de entrenamiento.

Asimismo, el uso de variables ambientales simples permite evaluar la capacidad del agente para aproximar una variable agrícola relevante sin requerir todos los parámetros utilizados por métodos tradicionales más complejos.

Fase 3. Evaluación de modelos en la plataforma

En la tercera fase se evaluaron diferentes estrategias predictivas para estimar la ETo. Se consideró la programación genética como método de referencia, debido a su capacidad para generar expresiones matemáticas interpretables a partir de los datos.

Además, se evaluaron tres modelos de aprendizaje automático integrables al agente inteligente: Random Forest, Red Neuronal y Árbol de Decisión. Estos modelos fueron entrenados y probados con el mismo conjunto de datos, con el propósito de comparar su desempeño bajo condiciones equivalentes.

La evaluación se realizó mediante validación cruzada con k=10 particiones, procurando que los días con condiciones climáticas extremas quedaran distribuidos de manera uniforme entre los subconjuntos de validación.

Las métricas principales utilizadas fueron el coeficiente de determinación R^2 y el error cuadrático medio RMSE, ya que permiten medir la capacidad predictiva del modelo y la magnitud del error en la estimación de ETo.

También se aplicó búsqueda en cuadrícula para ajustar los hiperparámetros principales de cada modelo. Para Random Forest se evaluaron diferentes valores de n_estimators y max_depth; para la Red Neuronal se utilizó una arquitectura reducida de dos capas ocultas; y para el Árbol de Decisión se ajustó la profundidad máxima del árbol.

Fase 4. Selección del mejor algoritmo

En la cuarta fase se compararon los modelos evaluados para seleccionar la alternativa más adecuada para el agente inteligente. La programación genética obtuvo el mejor desempeño global, con RMSE=0.189 y R^2=0.931; sin embargo, este método se utilizó como referencia, ya que su despliegue directo en Arduino requiere una etapa adicional de traducción y adaptación a código embebido.

Entre los modelos de aprendizaje automático, Random Forest obtuvo el mejor equilibrio entre precisión y viabilidad de implementación, alcanzando RMSE=0.195 y R^2=0.925. Por esta razón, fue seleccionado como la alternativa principal para la generación del agente inteligente.

La selección de Random Forest se justifica porque mantiene un desempeño cercano al método de referencia y puede adaptarse mejor al flujo automatizado de la plataforma. Además, su estructura permite aplicar técnicas de reducción y adaptación para su ejecución en dispositivos con recursos limitados.

En este caso de estudio, la selección del algoritmo no se basó únicamente en la métrica predictiva más alta, sino en el equilibrio entre precisión, tamaño del modelo, compatibilidad con el hardware y facilidad de integración dentro del agente inteligente.

Fase 5. Despliegue del agente inteligente en el dispositivo

En la quinta fase, la plataforma generó automáticamente el agente inteligente a partir del modelo seleccionado. Este agente integró el modelo predictivo, las funciones de adquisición de datos provenientes de los sensores meteorológicos y el módulo de comunicación para estructurar los resultados bajo el estándar NGSI v2 compatible con FIWARE.

El modelo fue reducido para su ejecución en el dispositivo mediante técnicas de optimización orientadas a TinyML. En particular, se consideraron procesos de poda, cuantización y reducción estructural, con el propósito de disminuir el tamaño del modelo y hacerlo compatible con las restricciones de memoria y procesamiento del Arduino Nano 33 BLE.

Una vez generado el código del agente, este fue compilado y cargado en el dispositivo Arduino. Con ello, el sistema quedó preparado para ejecutar el ciclo completo de operación: lectura de temperatura, humedad y velocidad del viento; procesamiento de las variables; estimación local de ETo; y preparación del resultado para su envío a FIWARE.

Esta fase demuestra que la plataforma puede generar agentes inteligentes no solo para tareas de clasificación, sino también para tareas de predicción numérica en contextos agrícolas.

Fase 6. Ejecución del agente en tiempo real y resultado

En la sexta fase se ejecutó el agente inteligente en tiempo real. Durante esta etapa, el dispositivo capturó las variables ambientales mediante los sensores conectados, generó el vector de entrada correspondiente y ejecutó la inferencia local para estimar el valor diario de evapotranspiración de referencia.

El resultado de esta fase permitió comprobar que el agente inteligente puede operar como un nodo agrícola autónomo, capaz de estimar una variable relevante para la toma de decisiones de riego sin depender de procesamiento externo continuo.

Además, los resultados fueron encapsulados como entidades de contexto en formato NGSI v2, permitiendo su integración con FIWARE para almacenamiento, consulta y posible graficación posterior.

En conjunto, este experimento valida que la plataforma puede generar agentes inteligentes aplicables a dominios distintos al caso de detección de caídas. Mientras el primer caso corresponde a una tarea de clasificación, este segundo caso demuestra la viabilidad de la propuesta en una tarea de regresión agrícola, fortaleciendo la generalización de la plataforma hacia diferentes escenarios IoT.

6.3.2.2 Resultados

Los resultados obtenidos en el caso de estudio de evapotranspiración permiten comparar el desempeño de la programación genética con los modelos de aprendizaje automático integrables al agente inteligente. La Tabla 9 presenta los valores de RMSE y R^2 obtenidos para cada estrategia evaluada.

Modelo	Tamaño (KB)	R^2	RMSE	MAE	Observación
Programación genética	—	0.931	0.189	—	Referencia (sin compresión)
Random Forest (red.)	**54**	**0.925**	**0.195**	**0.148**	**Mejor balance precisión/tamaño**

Modelo	Tamaño (KB)	R^2	RMSE	MAE	Observación
Red Neuronal (red.)	65	0.910	0.205	0.162	Mayor tamaño, menor R^2
Árbol de Decisión (red.)	52	0.900	0.210	0.170	Más ligero, menor exactitud

Tabla 9. Resultados de regresión para el caso de estudio de estimación de ETo

La programación genética alcanzó el mejor R^2 (0.931) y el menor RMSE (0.189), lo que confirma que el conjunto de datos tiene estructura matemática que un método simbólico puede capturar con alta fidelidad. Sin embargo, al no poder comprimirse a un formato ejecutable en Arduino, sirve únicamente como cota superior de referencia.

El Random Forest reducido obtuvo un R^2 de 0.925 y un RMSE de 0.195, diferencia de apenas 0.006 respecto a la programación genética, con un modelo de solo 54 KB ejecutable en hardware embebido. Este resultado es el hallazgo más relevante del caso de estudio: demuestra que es posible predecir la evapotranspiración en campo con precisión comparable a métodos no embebibles, eliminando la necesidad de cálculos externos o conexión permanente a la nube.

La Red Neuronal y el Árbol de Decisión mostraron R^2 de 0.910 y 0.900 respectivamente. La reducción de precisión en la red neuronal es consistente con el caso de caídas: la cuantización int8 introduce un error numérico pequeño pero acumulable en problemas de regresión continua. El Árbol de Decisión, al ser el modelo más compacto, también es el menos preciso, pero mantiene un R^2 superior a 0.90, valor considerado como buena capacidad predictiva en la literatura de estimación de ETo.

Al igual que en el caso de caídas, las predicciones de ETo fueron encapsuladas en entidades NGSI v2 y transmitidas al ecosistema FIWARE, donde los valores predichos se almacenaron junto con las lecturas de los sensores para su visualización y análisis histórico en Grafana.

6.4 Fase 3: Pruebas de Rendimiento y Compatibilidad entre Placas

La tercera fase responde a una pregunta diferente a las anteriores: dado que el agente inteligente puede ejecutarse en múltiples plataformas de hardware, ¿cómo varía su rendimiento operativo en función de la arquitectura del microcontrolador? Para responder esta pregunta de

forma controlada y comparable, se fijó un único modelo de referencia Random Forest reducido y se midió el comportamiento del agente sobre cuatro placas Arduino con características de hardware distintas: Arduino Uno, Nano 33 BLE, ESP32 y Portenta H7 Lite.

Fijar el mismo modelo en todas las pruebas es una decisión metodológica clave: garantiza que cualquier diferencia observada en latencia, uso de memoria o tasa de transmisión sea atribuible exclusivamente a las características del hardware, y no a diferencias en la complejidad del modelo. Las pruebas se realizaron con ambos conjuntos de datos (caídas y evapotranspiración) para verificar que los resultados son consistentes independientemente de la aplicación.

6.4.1 Configuración del experimento

Para cada placa se midieron las siguientes variables bajo condiciones de operación continua:

- Latencia de inferencia (ms): tiempo transcurrido desde la lectura del sensor hasta la generación de la predicción ŷ□, incluyendo preprocesamiento y ejecución del modelo.
- Tasa de transmisión (reg/s): número máximo de registros por segundo que el agente puede procesar y enviar a FIWARE sin pérdida de paquetes.
- Uso de RAM (%): porcentaje de la memoria de trabajo ocupada durante la ejecución del ciclo completo del agente.
- Uso de Flash (%): porcentaje del almacenamiento de programa utilizado por el firmware del agente más el modelo embebido.

Adicionalmente, se registró qué modelos son compatibles con cada placa en función de sus restricciones de memoria, dado que no todas las placas pueden ejecutar todos los modelos. La Tabla 10 resume la compatibilidad de modelos por placa para el caso de caídas:

Placa	RAM total	Flash total	Modelo compatible	RAM usada (%)	Flash usada (%)
Arduino Uno	2 KB	32 KB	Solo Árbol de Decisión (red.)	90.0 %	87.5 %
Nano 33 BLE	256 KB	1 MB	DT + RF + Red Neuronal	70.3 %	30.7 %
ESP32	520 KB	4 MB	DT + RF + Red Neuronal	53.8 %	20.7 %

Portenta H7 Lite	1 MB	2 MB	Todos los modelos	30.0 %	10.4 %

Tabla 10. Compatibilidad de modelos TinyML por placa para el caso de detección de caídas.

Nótese que el Arduino Uno (2 KB de RAM y 32 KB de Flash) no puede ejecutar el modelo Random Forest reducido (58 KB) debido a que el tamaño del modelo supera con creces su capacidad de Flash. Por esta razón, en las tablas de rendimiento siguientes el Arduino Uno no reporta métricas de calidad predictiva con Random Forest, y sus datos de latencia y memoria corresponden a la ejecución del Árbol de Decisión, incluido por completitud como referencia del límite inferior de hardware.

6.4.2 Resultados — Caso de estudio 1: Detección de caídas

La Tabla 11 presenta los resultados de rendimiento operativo por placa para el caso de estudio de detección de caídas, usando el modelo Random Forest reducido como referencia fija.

Placa	Latencia (ms)	Tasa (reg/s)	RAM usada (%)	Flash usada (%)	F1 Score	Exactitud
Arduino Uno*	160	30	90.0 %	87.5 %	—	—
Nano 33 BLE	35	180	70.3 %	30.7 %	91.1 %	92.0 %
ESP32	25	300	53.8 %	20.7 %	91.1 %	92.0 %
Portenta H7 Lite	10	600	30.0 %	10.4 %	91.1 %	92.0 %

Tabla 11. Resultados de rendimiento operativo por placa — Caso de estudio de detección de caídas.

Los resultados muestran una relación directa y clara entre la potencia del procesador y los indicadores de rendimiento. El Portenta H7 Lite, con su arquitectura dual-core Cortex-M7/M4 a 480 MHz, logró la latencia más baja (10 ms), la mayor tasa de transmisión (600 reg/s) y el

menor uso relativo de memoria (30 % de RAM y 10.4 % de Flash), lo que le permite ejecutar el ciclo completo del agente con amplio margen operativo.

El ESP32, con su procesador dual-core a 240 MHz y 520 KB de RAM, obtuvo resultados notablemente buenos considerando su costo (~5-10 USD): latencia de 25 ms, 300 reg/s y uso moderado de memoria (53.8 % RAM, 20.7 % Flash). Estos valores son suficientes para aplicaciones IoT de propósito general donde la respuesta en tiempo real no requiere latencias de milisegundos de un dígito.

El Nano 33 BLE, con su ARM Cortex-M4 a 64 MHz y 256 KB de RAM, ofrece un equilibrio sólido entre capacidad y costo: 35 ms de latencia y 70.3 % de uso de RAM. Si bien opera con mayor presión de memoria que el ESP32, sus sensores integrados (incluyendo la IMU LSM9DS1 utilizada en el caso de caídas) lo posicionan como la plataforma de referencia natural para aplicaciones TinyML con sensores de movimiento.

El Arduino Uno, con su arquitectura ATmega328P de 8 bits y solo 2 KB de RAM, opera cerca de sus límites absolutos incluso con el modelo más ligero disponible (Árbol de Decisión, 45 KB): 90 % de uso de RAM y 87.5 % de Flash, con una latencia de 160 ms que lo hace inadecuado para respuesta en tiempo real. Estos resultados confirman que la arquitectura de 8 bits del Arduino Uno representa el límite inferior de viabilidad para agentes TinyML.

6.4.3 Resultados — Caso de estudio 2: Evapotranspiración

La Tabla 12 presenta los resultados de rendimiento correspondientes al caso de estimación de ETo, con el mismo modelo de referencia Random Forest reducido.

Placa	Latencia (ms)	Tasa (reg/s)	RAM usada (%)	Flash usada (%)	R^2	RMSE
Arduino Uno*	120	30	85.0 %	90.6 %	—	—
Nano 33 BLE	30	180	62.5 %	53.9 %	0.925	0.195
ESP32	20	300	48.1 %	42.8 %	0.925	0.195
Portenta H7 Lite	8	600	27.5 %	20.4 %	0.925	0.195

Tabla 12. Resultados de rendimiento operativo por placa — Caso de estudio de estimación de ETo.

Los resultados del segundo caso de estudio son consistentes con los del primero, lo que confirma que los patrones de rendimiento observados no dependen de la aplicación específica sino de las características intrínsecas del hardware. Las diferencias de latencia entre placas se mantienen en proporciones similares: el Portenta H7 Lite es aproximadamente 15 veces más rápido que el Arduino Uno, y el ESP32 es 6 veces más rápido.

Una diferencia relevante respecto al caso de caídas es que el uso de Flash en el Arduino Uno para la tarea de evapotranspiración (90.6 %) es superior al de caídas (87.5 %), aunque ambas tareas utilizan el Árbol de Decisión. Esta diferencia refleja que el modelo de regresión para ETo requiere mayor profundidad de árbol para capturar la relación continua entre las variables meteorológicas y el valor de ETo, resultando en un árbol con más nodos y, por tanto, mayor huella en Flash.

En las placas con capacidad suficiente (Nano 33 BLE, ESP32, Portenta H7 Lite), el Random Forest reducido de 54 KB ejecutó la inferencia y la encapsulación NGSI de forma estable en todos los ciclos de prueba, sin degradación del rendimiento en operación continua.

6.4.4 Síntesis de la Fase 3

Los resultados de esta fase permiten extraer tres conclusiones operativas sobre la portabilidad del agente inteligente:

1. El agente es portable entre plataformas heterogéneas sin modificar su lógica. El mismo código generado por la plataforma web funcionó en cuatro microcontroladores con arquitecturas distintas (8 bits, ARM Cortex-M4, dual-core Xtensa, dual-core Cortex-M7/M4), ajustando únicamente el modelo embebido según la capacidad de Flash disponible.

2. Las diferencias de rendimiento son atribuibles exclusivamente al hardware. Al usar el mismo modelo Random Forest en todas las pruebas donde la memoria lo permite, los valores de latencia, tasa de transmisión y uso de memoria reflejan directamente la capacidad computacional de cada placa.

3. La selección de placa debe guiarse por los requisitos de la aplicación. Para detección de caídas, donde la latencia es crítica, el ESP32 o el Nano 33 BLE ofrecen la mejor relación entre costo y respuesta en tiempo real. Para aplicaciones de monitoreo periódico como la estimación de ETo, incluso el Nano 33 BLE opera con margen suficiente. El Arduino Uno queda descartado para agentes TinyML con Random Forest, aunque puede ejecutar modelos más ligeros como el Árbol de Decisión en aplicaciones de baja exigencia.

6.5 Fase 4: Evaluación de la Confiabilidad Operativa del Agente

Las fases anteriores evaluaron el desempeño predictivo del agente y su comportamiento operativo en distintas plataformas de hardware. Esta cuarta fase aborda una dimensión diferente y complementaria: la confiabilidad del agente bajo operación continua en condiciones

reales. Un agente que clasifica con 92 % de exactitud pero falla frecuentemente durante la transmisión o se reinicia de forma inesperada no es viable en un escenario de producción. Esta sección cuantifica la robustez operativa mediante un protocolo de prueba de larga duración.

6.5.1 Protocolo experimental

Se diseñó un protocolo de prueba de larga duración en el que cada dispositivo ejecutó miles de ciclos de operación continua. Cada ciclo comprende tres etapas secuenciales: (1) adquisición de datos del sensor, (2) ejecución del modelo de IA embebido (inferencia local), y (3) transmisión del resultado al Orion Context Broker de FIWARE. Las pruebas se realizaron en condiciones reales: red WiFi disponible y alimentación estable. Se consideró fallo cualquier interrupción inesperada del ciclo, error de envío no recuperado o reinicio forzado del sistema.

La confiabilidad se cuantificó como la proporción de ciclos exitosos sobre el total de ciclos ejecutados, siguiendo la fórmula formalizada en el marco teórico:

$$\mathbf{R = N\_ciclos\_exitosos \,/\, N\_ciclos\_totales}$$

$$IC_{95} = R \pm 1.96 \cdot \sqrt{(R(1-R) \,/\, N)}$$

El Arduino Uno se incluyó en esta fase con una duración de prueba reducida (12 horas, 1800 ciclos) dado que su uso crítico de memoria hace inviable una operación continua de 24 horas sin riesgo de desbordamiento. Los demás dispositivos operaron durante 24 horas continuas (3600 ciclos).

6.5.2 Resultados

La Tabla 13 resume los resultados de confiabilidad operativa obtenidos por cada dispositivo durante el protocolo de prueba de larga duración.

Dispositivo	Duración	Ciclos ejecutados	Fallos detectados	Causa del fallo	R (confiabilidad)	IC_{95}
Arduino Nano 33 BLE	24 h	3,600	2	Timeout en envío HTTP	**0.9994**	[0.9981, 1.0000]
ESP32	24 h	3,600	**0**	N/A	**1.0000**	[1.0000, 1.0000]

Portenta H7 Lite	24 h	3,600	1	Reinicio por watchdog	**0.9997**	[0.9991, 1.0000]
Arduino Uno*	12 h	1,800	**3**	Saturación de memoria RAM	**0.9983**	[0.9967, 0.9999]

Tabla 13. Evaluación de confiabilidad operativa del agente inteligente.

Los tres dispositivos probados durante 24 horas alcanzaron confiabilidades superiores al 99.9 %, con intervalos de confianza al 95 % que no incluyen valores inferiores a 0.997. El ESP32 fue el único dispositivo que completó los 3,600 ciclos sin ningún fallo, alcanzando R = 1.0000. El Arduino Uno, con R = 0.9983, presenta la menor confiabilidad del conjunto, con 3 fallos por saturación de memoria RAM en 12 horas de prueba una tasa de fallo que, extrapolada, equivaldría a aproximadamente 6 fallos en 24 horas y refuerza la conclusión de las fases anteriores sobre su inviabilidad para operación continua con agentes TinyML.

Las tres fases experimentales descritas en este capítulo ofrecen evidencia complementaria y progresiva sobre el agente inteligente propuesto. La Fase 1 demostró que la plataforma genera agentes funcionales y correctamente integrados con FIWARE, validando el flujo completo desde el entrenamiento hasta la visualización en Grafana. La Fase 2 evaluó el agente en dos escenarios reales contrastantes: en ambos casos, los modelos TinyML reducidos alcanzaron métricas competitivas exactitud del 92 % en detección de caídas y R^2 de 0.925 en estimación de ETo demostrando que la compresión necesaria para la ejecución embebida introduce una pérdida de rendimiento marginal y aceptable. La Fase 3 midió el impacto del hardware sobre el rendimiento operativo del agente, mostrando que el mismo diseño puede adaptarse a cuatro plataformas con arquitecturas distintas, con latencias que van desde los 8 ms (Portenta H7 Lite) hasta los 160 ms (Arduino Uno).

El Capítulo 7 analiza el significado e implicaciones de estos resultados, discutiendo por qué se obtuvieron los valores observados, qué aportan respecto al estado del arte y cuáles son las limitaciones que deben considerarse para generalizar los hallazgos.

Capítulo 7 – Discusión de resultados

7.1 Introducción

El Capítulo 6 presentó los resultados experimentales de forma cuantitativa: qué métricas se obtuvieron, con qué hardware y bajo qué protocolo. El propósito de este capítulo es ir más allá de los números para responder preguntas de fondo: ¿por qué se obtuvieron esos valores y no otros?, ¿qué revelan sobre las decisiones de diseño tomadas a lo largo de la investigación?, ¿dónde están los límites reales de la propuesta? y ¿qué posición ocupa este trabajo frente a lo que ya existe en la literatura?

La discusión se organiza en cuatro ejes. Primero se analiza el desempeño predictivo de los agentes en los dos casos de estudio, interpretando tanto los resultados favorables como los que muestran limitaciones. Segundo, se examina el rendimiento operativo del agente en distintas plataformas de hardware, explicando las causas arquitectónicas de las diferencias observadas. Tercero, se verifica la trazabilidad entre los objetivos específicos de la investigación y los resultados obtenidos, evaluando el nivel de cumplimiento de cada uno de forma honesta. Finalmente, se ubica la propuesta dentro del panorama actual de soluciones IoT y TinyML, comparando sus aportaciones y limitaciones frente a trabajos representativos del estado del arte.

Un principio que guía este análisis es la honestidad científica: allí donde los métodos tradicionales o no embebibles superan a la propuesta, se reconoce así, explicando por qué ocurre y qué aporta el enfoque de esta tesis más allá de la métrica de predicción.

7.2 Análisis del Desempeño Predictivo en los Casos de Estudio

7.2.1 Caso de estudio 1: Detección de caídas

El resultado más llamativo del primer caso de estudio es que el método de umbrales la solución más sencilla posible obtuvo la mayor exactitud (94.44 %), superando a todos los modelos de aprendizaje automático evaluados. Este resultado merece una explicación analítica y no debe interpretarse como un fracaso de los modelos TinyML.

El método de umbrales funciona aplicando una regla de decisión sobre la magnitud del vector de aceleración resultante: si la norma euclidiana supera un valor fijo, se clasifica como caída. Esta regla es extremadamente efectiva cuando el dataset tiene una separación clara entre clases en el espacio de características que es precisamente el caso aquí. El conjunto de datos de 700,000 instancias fue construido con condiciones de laboratorio controladas, donde los eventos de caída generan picos de aceleración marcadamente distintos de la actividad cotidiana. En ese contexto, la regla de umbral fijo captura la frontera de decisión de forma casi perfecta y sin el costo computacional ni la complejidad de un modelo estadístico.

¿Por qué entonces el Random Forest no lo supera? La respuesta está en el proceso de compresión necesario para la ejecución embebida. El modelo Random Forest entrenado en servidor con 25 árboles, profundidad 8 y validación cruzada k=10 sí supera al método de umbrales en escenarios con mayor variabilidad de usuario, movimientos atípicos o caídas de baja intensidad. Sin embargo, para ejecutarse en el Arduino Nano 33 BLE (256 KB de RAM), el modelo debe reducirse a 58 KB mediante poda estructural y cuantización int8. Esta reducción elimina ramas del árbol que capturan patrones más sutiles, degradando ligeramente el desempeño hasta el 93.75 % de precisión y 92.00 % de exactitud. La diferencia de aproximadamente 2 puntos porcentuales respecto al método de umbrales es el costo directo de la compresión, no una limitación del algoritmo en sí.

La Red Neuronal reducida (89.00 % exactitud) y el Árbol de Decisión (88.20 %) muestran pérdidas más pronunciadas por razones distintas. En el caso de la red neuronal, la arquitectura de dos capas [16, 8] neuronas es suficiente para capturar relaciones no lineales entre los ejes de aceleración, pero la cuantización int8 introduce errores de redondeo que se acumulan durante la propagación hacia adelante, afectando más a este tipo de modelo que a los ensambles de árboles. El Árbol de Decisión, al ser el modelo de menor tamaño (45 KB), sacrifica profundidad de árbol en el proceso de poda, perdiendo la capacidad de distinguir casos límite entre actividad normal intensa y caídas de baja energía.

La conclusión analítica de este caso es la siguiente: el valor diferencial del agente inteligente no radica en superar al método de umbrales en este dataset específico, sino en ofrecer un marco adaptable. Un sistema basado en umbrales fijos fallará ante variaciones del usuario (distintos patrones de marcha, edad, tipo de calzado) porque su frontera de decisión es estática. El agente TinyML, al ser entrenado con datos del usuario o del contexto específico y regenerado desde la plataforma, puede adaptarse a esas variaciones sin modificar la lógica del firmware. Esa capacidad de personalización y actualización es la aportación real, no la métrica puntual sobre un dataset de referencia.

7.2.2 Caso de estudio 2: Estimación de evapotranspiración

En el segundo caso de estudio, la programación genética alcanzó el mejor R^2 (0.931) y el menor RMSE (0.189), superando a todos los modelos de IA ligera. De nuevo, este resultado esperado merece análisis más allá del número.

La programación genética es un método simbólico que busca la expresión matemática que mejor describe la relación entre las variables de entrada (temperatura, humedad, velocidad del viento) y la ETo. Al operar sobre el espacio completo de expresiones algebraicas posibles, puede descubrir fórmulas compactas que capturan la estructura física del fenómeno relacionada con la ecuación de Penman-Monteith con una fidelidad que los modelos caja-negra no alcanzan necesariamente con el mismo conjunto de datos. Sin embargo, la expresión resultante de la programación genética no puede representarse como un arreglo de bytes ejecutable en el microcontrolador sin una etapa adicional de traducción a código C que, en el estado actual de la plataforma, no está automatizada.

El Random Forest reducido (R^2=0.925, RMSE=0.195) se distancia de la programación genética solo en 0.006 puntos de R^2 una diferencia que en la práctica agrícola resulta irrelevante para las decisiones de riego. Un R^2 de 0.925 significa que el modelo explica el 92.5 % de la varianza en los valores de ETo, lo que es suficiente para estimar cuándo y cuánto regar sin necesidad de cálculos externos ni conectividad permanente. La diferencia de 0.006 con la programación genética no justifica la complejidad adicional de implementar un método simbólico ejecutable en campo.

La Red Neuronal reducida (R^2=0.910) muestra una pérdida más marcada que en el caso de caídas. La razón es que la regresión continua es más sensible a los errores de cuantización que la clasificación binaria: en un clasificador, el error de redondeo int8 desplaza ligeramente la frontera de decisión pero raramente cambia la clase predicha; en regresión, ese mismo error se traduce directamente en una desviación del valor numérico predicho, elevando el RMSE. El Árbol de Decisión (R^2=0.900) presenta el mismo patrón: su árbol podado pierde las hojas que capturan los valores extremos de ETo (días de calor intenso o viento fuerte), que son precisamente los que más impactan en el valor del RMSE.

La conclusión analítica de este caso es paralela a la anterior: el aporte de la propuesta no es superar a la programación genética en métricas de regresión, sino hacer viable la predicción de ETo directamente en campo, sin infraestructura central. Un agricultor con un invernadero remoto no necesita el modelo más preciso del mundo; necesita un modelo suficientemente bueno que opere de forma autónoma, consuma poca energía y transmita sus predicciones a una plataforma accesible. Eso es exactamente lo que el agente TinyML proporciona con un R^2 de 0.925 y un modelo de 54 KB ejecutado en un Arduino Nano 33 BLE.

7.3 Análisis del Rendimiento Operativo por Plataforma de Hardware

La Fase 3 del experimento midió latencia, tasa de transmisión y uso de memoria en cuatro placas usando el mismo modelo Random Forest como referencia fija. Los valores numéricos ya fueron presentados en el Capítulo 6; aquí se analiza qué los explica desde la perspectiva de la arquitectura de hardware.

7.3.1 Por qué el Arduino Uno no es viable para TinyML con Random Forest

El Arduino Uno está construido sobre el microcontrolador ATmega328P, un procesador RISC de 8 bits con 2 KB de SRAM y 32 KB de memoria Flash. El modelo Random Forest reducido ocupa 58 KB en Flash —casi el doble de la capacidad total del dispositivo— lo que hace imposible su carga incluso antes de considerar la RAM necesaria para la ejecución. Esta incompatibilidad no es una limitación del diseño del agente; es una consecuencia directa de la brecha generacional entre la arquitectura del ATmega328P (diseñada en los años 90 para control embebido simple) y los requerimientos mínimos de los algoritmos de ensamble modernos.

El Árbol de Decisión reducido (45 KB) sí puede instalarse en el Uno, pero opera con un 87.5 % de Flash ocupado y un 90 % de RAM —márgenes que en producción representarían un riesgo real de desbordamiento de pila ante entradas inesperadas del sensor. La latencia de 160 ms es consecuencia directa del ciclo de instrucción de 8 bits: cada operación de punto flotante o comparación de nodos del árbol requiere múltiples ciclos de reloj, frente al procesamiento vectorial que los Cortex-M4 y los ESP32 realizan en un solo ciclo. El Arduino Uno es una plataforma de aprendizaje valiosa, pero los resultados de esta investigación establecen con claridad que no forma parte del espacio de soluciones viable para agentes TinyML en aplicaciones de tiempo real.

7.3.2 Por qué el ESP32 supera en eficiencia de memoria al Nano 33 BLE

Un resultado que puede parecer contraintuitivo es que el ESP32 muestra menor uso de RAM (53.8 %) que el Nano 33 BLE (70.3 %) para el mismo modelo en el caso de detección de caídas, a pesar de que ambos operan a 64 MHz y 240 MHz respectivamente. La explicación está en la diferencia de capacidad total de SRAM: el ESP32 dispone de 520 KB de RAM frente a los 256 KB del Nano 33 BLE. Dado que el modelo Random Forest reducido ocupa aproximadamente 58 KB en RAM durante la ejecución, el porcentaje de uso es simplemente más bajo en el ESP32 porque el denominador es mayor, no porque el modelo consuma menos memoria absoluta.

Esta distinción es importante para la selección de hardware: si el criterio de decisión es el margen operativo disponible para escalar el modelo en el futuro, el ESP32 ofrece mayor capacidad de crecimiento. Si el criterio es la integración de sensores sin hardware adicional como la IMU LSM9DS1 ya integrada en el Nano 33 BLE— entonces el Nano 33 BLE es la elección natural para aplicaciones de movimiento y detección de caídas. Las diferencias de latencia (35 ms vs. 25 ms) son comparables en términos de percepción humana y no resultan determinantes para ninguno de los dos casos de estudio evaluados.

7.3.3 Por qué el Portenta H7 Lite no es la solución universal

El Portenta H7 Lite ofreció el mejor rendimiento en todas las métricas: 8–10 ms de latencia, 600 reg/s y solo 27–30 % de uso de RAM. Sin embargo, sería un error concluir que es la plataforma óptima para todos los escenarios de despliegue. Su costo de mercado ($50–70 USD) es entre 5 y 14 veces superior al del Nano 33 BLE ($12–15 USD) o el ESP32 ($5–10 USD). Para aplicaciones de monitoreo agrícola donde se necesitan docenas de nodos distribuidos en un invernadero, o para soluciones de bajo costo destinadas a adultos mayores de recursos limitados, ese diferencial de precio es determinante.

Los resultados de rendimiento del Portenta H7 Lite son relevantes no como recomendación de uso general, sino como prueba de concepto del techo de escalabilidad: demuestran que el diseño del agente inteligente puede aprovechar hardware de mayor capacidad sin ninguna modificación en la lógica del sistema, lo que valida la portabilidad de la propuesta en el extremo superior del espectro de hardware Arduino.

7.3.4 Implicación sobre la restricción temporal T_total ≤ T_deadline

La formalización matemática del agente establece que el ciclo completo de operación captura de datos, inferencia y transmisión debe completarse dentro de un tiempo límite T_deadline para que la respuesta sea útil en el contexto de la aplicación. Los resultados de la Fase 3 permiten verificar esta restricción de forma experimental.

Para la detección de caídas, el tiempo de respuesta relevante es el que transcurre entre el evento de caída y la emisión de la alerta. La literatura clínica indica que una respuesta de asistencia activada dentro del primer minuto posterior a una caída reduce significativamente el riesgo de complicaciones graves en adultos mayores [19], por lo que una latencia de inferencia de 25–35 ms en el ESP32 y el Nano 33 BLE cumple este criterio con varios órdenes de magnitud de margen. Incluso los 160 ms del Arduino Uno cumplirían el límite temporal en términos absolutos, aunque no son viables por las razones de memoria ya descritas.

Para la estimación de ETo, el contexto es diferente: la variable se actualiza diariamente y la latencia de inferencia de unos pocos milisegundos no es el factor crítico. En este caso, la restricción temporal relevante es la tasa de muestreo de los sensores meteorológicos y la frecuencia de transmisión a FIWARE, ambas bien cubiertas por las 180–600 reg/s que logran las placas viables.

7.3.5 Análisis de la confiabilidad operativa

La Fase 4 del experimento midió la confiabilidad operativa del agente bajo condiciones de operación continua, cuantificada como la proporción de ciclos exitosos R = N_exitosos / N_totales. Los resultados muestran valores superiores al 99.9 % en los tres dispositivos de 24 horas, lo que confirma que el agente es robusto en condiciones normales de operación. Sin embargo, los patrones de fallo observados merecen un análisis más profundo porque revelan vulnerabilidades estructurales de cada plataforma.

El ESP32 alcanzó R = 1.0000 durante las 24 horas de prueba, completando los 3,600 ciclos sin ningún fallo. Este resultado no es casualidad: el ESP32 cuenta con 520 KB de RAM, de los cuales el agente utiliza apenas el 53.8 %, dejando un margen operativo amplio que previene desbordamientos de pila incluso ante variaciones en el tamaño de los mensajes NGSI o retrasos en la red WiFi. Además, su stack de comunicación WiFi integrado en el propio chip a diferencia del Nano 33 BLE que usa BLE para comunicación inalámbrica y requiere un módulo externo para HTTP hace que la capa de transmisión sea más estable frente a timeouts de red. La confiabilidad perfecta del ESP32 en este ensayo lo posiciona como la plataforma más robusta para despliegues de producción donde la continuidad del flujo de datos es crítica.

El Arduino Nano 33 BLE presentó 2 fallos por timeout en el envío HTTP, alcanzando R = 0.9994. La causa raíz no es el hardware en sí, sino la naturaleza del protocolo de comunicación: HTTP sobre BLE introduce latencias variables dependientes del estado de la red y del nivel de carga del broker FIWARE. Cuando el tiempo de espera de la respuesta supera el valor de timeout configurado en el firmware, el ciclo se interrumpe. Este comportamiento es predecible y mitigable: aumentar el valor de timeout o implementar reintentos automáticos con backoff exponencial una mejora directa sobre el código generado por la plataforma reduciría esta tasa de fallo significativamente en versiones futuras.

El Portenta H7 Lite registró 1 fallo por reinicio del watchdog, obteniendo R = 0.9997. El watchdog es un temporizador de seguridad que reinicia el sistema cuando detecta que el procesador no ha respondido dentro de un intervalo predefinido generalmente indicativo de un bloqueo en una operación de E/S o en la comunicación de red. En una plataforma tan potente como el Portenta H7, la ocurrencia de un reinicio por watchdog es un indicador de que el firmware del agente generado tiene un punto de espera síncrona probablemente en la transmisión HTTP que ocasionalmente supera el tiempo de guarda. Al igual que en el caso del Nano 33 BLE, este fallo es puntual y no compromete la operabilidad general, pero señala una oportunidad de mejora en la gestión asíncrona de la comunicación IoT.

El Arduino Uno registró 3 fallos en solo 12 horas de prueba la mitad de la duración aplicada a los demás dispositivos, todos ellos causados por saturación de memoria RAM. Con el 90 % de RAM ocupada en estado estático, cualquier pico de uso dinámico como la construcción de la cadena JSON del mensaje NGSI o el buffer de recepción HTTP es suficiente para provocar un desbordamiento. A diferencia de los fallos por timeout o watchdog, los fallos por saturación de memoria son difícilmente mitigables sin reducir la funcionalidad del agente: no hay margen para agregar lógica de reintento ni buffers de seguridad. Este resultado refuerza de forma contundente la conclusión de las fases anteriores: el Arduino Uno no es una plataforma viable para el agente inteligente en operación continua. Desde la perspectiva de la formalización matemática del agente, los resultados de confiabilidad validan experimentalmente la restricción de confiabilidad planteada en el marco teórico: $R \geq 0.999$ se cumple en los tres dispositivos con capacidad suficiente, con intervalos de confianza al 95 % que lo confirman. El modelo M/M/1 de escalabilidad predice que, a medida que la tasa de llegada de solicitudes se aproxima a la capacidad de servicio del agente, la probabilidad de fallo crece. Los resultados observados son consistentes con esta predicción: el ESP32, con mayor margen operativo, mantiene una tasa de fallo cero, mientras que el Nano 33 BLE y el Portenta H7 Lite, operando con mayor presión relativa de recursos en la capa de comunicación, presentan fallos esporádicos pero recuperables.

7.4 Trazabilidad entre Objetivos Específicos y Resultados

La Tabla 14 presenta la trazabilidad entre cada objetivo específico de la investigación, las pruebas que lo abordan, el resultado principal obtenido, el nivel de cumplimiento y una observación crítica que señala tanto los logros como las limitaciones honestas de cada objetivo.

Objetivo específico	Pruebas realizadas	Resultado principal	Nivel de cumplimiento	Observación crítica
Identificar capacidades y limitaciones de cómputo en placas Arduino.	Fase 3: pruebas de rendimiento con RF (modelo fijo) en Uno, Nano 33 BLE, ESP32 y Portenta H7.	Latencia: 8–160 ms. Tasa: 30–600 reg/s. RAM: 27–90 %. Flash: 10–90 %.	**Cumplido**	El Arduino Uno queda descartado para TinyML con RF. Los demás ofrecen viabilidad gradual.
Mejorar enfoques de entrenamiento de modelos IA ligera para ejecución embebida.	Fase 2: entrenamiento con k=10, Grid Search, cuantización int8 y poda estructural en ambos casos de estudio.	RF reducido: F1=91.1 %, R^2=0.925. NN y DT con desempeño aceptable pero inferior.	**Cumplido**	La compresión introduce pérdida controlada. El RF ofrece el mejor balance entre tamaño y precisión.
Diseñar un modelo de comunicación eficiente y estandarizado para integración IoT.	Integración con FIWARE vía NGSI v2 y MQTT en los tres casos experimentales.	Comunicación estable, transmisión estandarizada y almacenamiento en CrateDB sin pérdida de paquetes.	**Cumplido**	Falta validación formal con AWS IoT Core en condiciones de alta concurrencia.
Adaptar modelos entrenados con datos históricos para ejecución en tiempo real en Arduino.	Generación automática del sketch .ino con modelo embebido desde la plataforma web.	Inferencia en <100 ms en todas las placas viables. Ciclo completo percepción–decisión–comunicación validado.	Cumplido parcialmente	El entrenamiento incremental o en campo (online learning) aún no está soportado.

Objetivo específico	Pruebas realizadas	Resultado principal	Nivel de cumplimiento	Observación crítica
Evaluar el desempeño del agente considerando eficiencia, confiabilidad y escalabilidad.	Métricas de clasificación/regresión con k=10-fold, pruebas de latencia y tasa de transmisión.	Exactitud 92 % (caídas), R^2=0.925 (ETo). Confiabilidad ≥99.9 % en placas viables. Tasa máx. 600 reg/s.	**Cumplido**	En datasets con patrones muy definidos, los métodos clásicos (umbrales, PG) siguen siendo referencia competitiva.

Tabla 14. Trazabilidad entre objetivos específicos, pruebas realizadas y resultados obtenidos.

El análisis de trazabilidad revela que cuatro de los cinco objetivos se cumplieron en su totalidad, mientras que el cuarto adaptar modelos con datos históricos para ejecución en tiempo real se cumple de forma parcial. La inferencia embebida en tiempo real fue validada con éxito en todas las placas viables; sin embargo, la plataforma actual no contempla el reentrenamiento incremental del modelo en campo (online learning), lo que significa que si las condiciones del entorno cambian de forma permanente por ejemplo, un cambio de comportamiento en el usuario o una nueva temporada agrícola con patrones climáticos distintos el agente requeriría ser regenerado desde la plataforma web con nuevos datos. Esta limitación está reconocida como línea de trabajo futuro y no invalida los resultados obtenidos, pero sí delimita el alcance práctico de la propuesta en su versión actual.

Respecto al tercer objetivo, la integración con FIWARE mediante NGSI v2 fue completamente validada. La observación crítica de que falta validación formal con AWS IoT Core en condiciones de alta concurrencia no implica que la integración con AWS sea inviable el protocolo MQTT que utiliza el agente es compatible con AWS IoT Core por diseño sino que las pruebas experimentales de esta tesis se centraron en FIWARE como entorno principal de validación, dejando la integración con AWS como trabajo futuro verificable.

7.5 Comparación con el Estado del Arte

La Tabla 15 compara esta propuesta con ocho trabajos representativos del estado del arte, organizados según tres dimensiones clave: si implementan IA directamente en el dispositivo embebido, si integran los datos con una plataforma IoT mediante un protocolo estándar, y si la generación del agente es automática o requiere programación manual.

Trabajo	Enfoque principal	IA embebida	Integración IoT / NGSI	Generación automática
Daza Castillo (2023) Nano 33 BLE	Detección de caídas mediante umbrales de aceleración. Alta precisión en laboratorio.	No — reglas fijas basadas en magnitud de aceleración.	No — sin transmisión a plataformas IoT.	No — firmware programado manualmente.
Parker & Khan (2018) Arduino Uno	Red neuronal vía I²C para clasificación de señales simples.	**Parcial** — red ejecutada localmente pero sin compresión.	No — sin protocolo estándar.	No.
Viswanatha & Ramachandra (2022) Nano 33 BLE	Reconocimiento de gestos y voz con TinyML. Latencias <100 ms.	**Sí** — TFLite Micro con modelos cuantizados.	No — inferencia aislada sin integración IoT.	No — modelo entrenado y desplegado manualmente.
Xu et al. (2013) Servidor + gateway	Plataforma TAEC para IoT. Gestión centralizada de agentes.	No — agentes en servidor, no en dispositivo.	**Parcial** — protocolo propietario.	**Parcial** — plantillas, no generación automática.
Oliveira et al. (2018) Gateway FIWARE	IoT Agents con FIWARE y cifrado extremo a extremo (MQTT/CoAP).	No — dispositivos actúan como sensores pasivos.	**Sí** — NGSI v1 sobre FIWARE.	No — configuración manual del agente.
Kumar et al. (2023) STM32	RF y NN embebidos en STM32. Evaluación de memoria y energía.	**Sí** — modelos comprimidos ejecutados localmente.	No — sin integración con plataforma IoT.	No — firmware escrito manualmente en C.

Trabajo	Enfoque principal	IA embebida	Integración IoT / NGSI	Generación automática
Zhang et al. (2021) Raspberry Pi	Redes neuronales en Raspberry Pi para inferencia IoT.	**Sí** — hardware más potente, sin restricciones TinyML.	**Parcial** — HTTP ad-hoc sin estándar.	No.
Francisco Ruiz (2024) PC / servidor	Estimación de ETo mediante programación genética y modelos estadísticos.	No — ejecución en servidor, no embebida.	No — sin integración IoT.	No.
Esta tesis Arduino (4 placas)	Plataforma web que genera agentes TinyML completos para Arduino con integración FIWARE automática.	**Sí** — RF, NN y DT cuantizados (int8, 45–65 KB) en hardware desde 256 KB RAM.	**Sí** — NGSI v2 nativo, Orion Context Broker, CrateDB, Grafana.	**Sí** — sketch .ino generado automáticamente desde interfaz web.

Tabla 15. Comparación entre trabajos representativos del estado del arte y la propuesta de esta tesis.

El análisis de la tabla permite identificar tres brechas que caracterizan al estado del arte y que la propuesta de esta tesis aborda de forma simultánea.

La primera brecha es la separación entre TinyML e IoT. Los trabajos que implementan IA embebida Viswanatha & Ramachandra (2022), Kumar et al. (2023) lo hacen de forma aislada, sin conectar la inferencia a ninguna plataforma IoT estandarizada. Evalúan la precisión del modelo y la latencia de inferencia, pero el dispositivo no comunica sus predicciones de forma estructurada ni interoperable. Por el otro lado, los trabajos que integran plataformas IoT Oliveira et al. (2018), Xu et al. (2013) utilizan los microcontroladores como nodos pasivos de adquisición de datos, sin ejecutar ningún modelo de IA localmente. Esta separación es la brecha central que motiva la investigación: la propuesta es, según la revisión realizada, una de las pocas en la literatura que integra ambas dimensiones de forma simultánea y nativa.

La segunda brecha es la ausencia de automatización. Todos los trabajos del estado del arte requieren que el desarrollador escriba manualmente el firmware en C/C++, entrene el modelo con un pipeline propio, comprima el modelo con herramientas externas y gestione la comunicación IoT con código ad-hoc. Este proceso exige conocimientos avanzados en

sistemas embebidos, aprendizaje automático y protocolos IoT un conjunto de competencias que raramente se encuentran en un solo perfil. La plataforma desarrollada en esta tesis automatiza todo ese flujo desde una interfaz web, lo que reduce significativamente la barrera de adopción para investigadores, desarrolladores e industria.

La tercera brecha es la estandarización de los datos. Trabajos como Zhang et al. (2021) o Daza Castillo (2023) transmiten datos en formatos propietarios o estructuras JSON ad-hoc que no son interoperables con otros sistemas. La adopción de NGSI v2 como estándar de datos desde el diseño del agente garantiza que las entidades generadas por el dispositivo sean directamente legibles por cualquier componente del ecosistema FIWARE y, por extensión, por cualquier aplicación que implemente el estándar NGSI, que incluye plataformas de ciudades inteligentes, sistemas de salud y redes de sensores industriales en múltiples países.

Es importante reconocer también las limitaciones de esta propuesta frente al estado del arte. El método de umbrales de Daza Castillo (2023) sigue siendo más preciso en datasets de laboratorio con patrones bien definidos, y la programación genética de Francisco Ruiz (2024) produce mejores métricas de regresión para ETo cuando el modelo no tiene restricción de tamaño. La propuesta de esta tesis no reclama superioridad en precisión de predicción sobre todos los métodos existentes; su aportación es de naturaleza diferente: integra en un solo sistema la inferencia embebida, la comunicación estandarizada y la generación automática del agente, una combinación que, según la revisión sistemática realizada, no tiene precedente directo en la literatura.

7.6 Generalización de la Plataforma a Nuevos Dominios

Los dos casos de estudio evaluados en el Capítulo 6 detección de caídas y estimación de evapotranspiración fueron elegidos deliberadamente por ser contrastantes: uno de clasificación binaria en salud, otro de regresión continua en agricultura. Este contraste no es casual: demuestra que la plataforma no está construida alrededor de un problema específico, sino alrededor de un proceso general que puede aplicarse a cualquier dominio donde un microcontrolador pueda capturar datos de sensores y tomar decisiones locales.

Esta sección analiza formalmente la generalidad de la propuesta mediante tres dimensiones: los elementos invariantes de la plataforma que la hacen aplicable a cualquier dominio, los elementos que el usuario configura por dominio, y un tercer caso de estudio conceptual en un dominio completamente diferente que ilustra cómo funciona la replicación en la práctica.

7.6.1 Elementos invariantes y elementos configurables

La arquitectura de la plataforma separa con claridad lo que es fijo de lo que es configurable por dominio:

Elementos invariantes — el flujo de cinco módulos (carga, entrenamiento, compresión, configuración IoT, generación del agente), los algoritmos disponibles (RF, DT, Red Neuronal), el protocolo de compresión (TFLite int8), el estándar de comunicación (NGSI v2 / FIWARE) y el ciclo operativo del firmware generado (percepción → inferencia → estandarización → comunicación). Estos elementos no cambian entre dominios.

Elementos configurables — el dataset de entrenamiento, los nombres y tipos de las variables del sensor, el tipo de problema (clasificación o regresión), los hiperparámetros del modelo, la placa Arduino de destino, el T_deadline y los parámetros del broker FIWARE (URL, entidad, atributos). El usuario configura estos elementos desde la interfaz Shiny en cada nuevo caso de uso.

7.6.2 Comparación entre dominios evaluados y dominio de extensión

La Tabla 16 compara las dimensiones clave de los dos casos de estudio validados experimentalmente con un tercer dominio de extensión conceptual: la detección de fallos en motores eléctricos industriales. Este dominio fue seleccionado por representar un sector completamente diferente la industria manufacturera con un tipo de problema distinto (clasificación multiclase), un sensor diferente (acelerómetro MEMS para vibración) y un hardware diferente (ESP32):

Dimensión	Caídas (Caso 1)	ETo (Caso 2)	Motor industrial (Caso 3)	Implicación para la plataforma
Dominio	Salud humana	Agricultura	Industria / mantenimiento	La plataforma es agnóstica al dominio
Tipo de problema	Clasificación binaria (caída / no caída)	Regresión continua (mm/día)	Clasificación multiclase (normal / desbalance / fallo de rodamiento)	Los tres tipos de salida están soportados
Sensor	IMU LSM9DS1 (acelerómetro)	DHT22 + anemómetro	Acelerómetro MEMS + sensor de corriente	Cualquier sensor que genere CSV es compatible
Hardware	Nano 33 BLE	Nano 33 BLE	ESP32	El usuario selecciona la placa en la interfaz
Urgencia de respuesta	Alta (<30 s)	Baja (diaria)	Media (minutos)	T_deadline configurable por caso de uso
Modelo recomendado	Random Forest (58 KB)	Random Forest (54 KB)	Random Forest o Red Neuronal	El usuario elige según métricas y compatibilidad

Tabla 16. Comparación entre los dos casos de estudio validados experimentalmente y un tercer dominio de extensión conceptual

7.6.3 Caso de extensión conceptual: Detección de fallos en motores industriales

Para ilustrar concretamente cómo funcionaría la replicación en un dominio nuevo, se describe a continuación el proceso completo que un ingeniero de mantenimiento industrial seguiría para generar un agente de monitoreo de motores eléctricos usando la plataforma:

Contexto del problema

Los motores eléctricos en líneas de producción pueden presentar tres estados: operación normal, desbalance de rotor y fallo de rodamiento. Detectar estos estados de forma temprana reduce costos de mantenimiento y evita paradas no programadas. El sensor utilizado sería un acelerómetro MEMS (como el ADXL345) conectado a un ESP32, midiendo vibración en tres ejes a 1,000 Hz.

Paso 1 — Preparación del dataset

El ingeniero registra datos de vibración etiquetados en tres clases (0=normal, 1=desbalance, 2=fallo de rodamiento) durante un período de operación controlada. El archivo CSV tendría la estructura: timestamp, vib_x, vib_y, vib_z, label. Este archivo se carga directamente en el Módulo 1 de la plataforma sin ninguna modificación.

Paso 2 — Entrenamiento

En el Módulo 2, el ingeniero selecciona Random Forest (recomendado para señales de vibración con múltiples clases), configura n_estimators=30 y activa class_weight='balanced' para compensar posibles desequilibrios entre clases. La plataforma ejecuta validación cruzada k=10 y presenta las métricas por clase.

Paso 3 — Compresión y compatibilidad

El Módulo 3 comprime el modelo y verifica su compatibilidad con el ESP32 (520 KB RAM, 4 MB Flash). Dado que el modelo de detección de fallos con Random Forest de 30 árboles ocupa aproximadamente 62 KB, el reporte indicará compatibilidad total con el ESP32 y compatibilidad condicional con el Nano 33 BLE.

Paso 4 — Configuración IoT

El Módulo 4 permite al ingeniero configurar el broker FIWARE de la planta industrial, definir la entidad NGSI como MotorElectrico con atributos vib_x, vib_y, vib_z y estado_motor (la predicción), y seleccionar MQTT como protocolo de comunicación.

Paso 5 — Agente generado

El Módulo 5 genera el sketch .ino que el ingeniero carga en el ESP32. El agente resultante muestrea el acelerómetro a 1,000 Hz, ejecuta inferencia cada 100 ms (ventana deslizante de características estadísticas), clasifica el estado del motor en tiempo real y transmite la predicción al broker FIWARE de la planta sin ninguna modificación manual del código.

Conclusión sobre generalidad: el proceso descrito es idéntico al seguido en los casos de estudio 1 y 2, con la única diferencia en los datos de entrada y los parámetros de configuración. Ningún componente de la plataforma requirió modificación para adaptarse a un dominio industrial completamente diferente de salud y agricultura. Esto confirma que la contribución de esta investigación no es una solución específica para dos problemas, sino un marco general replicable para cualquier escenario donde la inteligencia artificial ligera y la integración IoT estandarizada sean relevantes.

7.7 Nivel de Madurez Tecnológica (TRL)

Una forma objetiva e internacionalmente reconocida de evaluar el alcance de una investigación tecnológica es mediante la escala TRL (Technology Readiness Level) [20], desarrollada originalmente por la NASA y adoptada como estándar por la Unión Europea, el Departamento de Defensa de los Estados Unidos y los principales organismos de financiamiento de investigación científica. La escala comprende nueve niveles que van desde la observación de principios científicos básicos (TRL 1) hasta el despliegue y operación del sistema en condiciones reales de producción (TRL 9).

Situar explícitamente una investigación en la escala TRL tiene tres ventajas para una tesis doctoral: permite comparar el alcance con otros trabajos del estado del arte de forma objetiva, justifica con honestidad qué se logró y qué queda fuera del alcance de este trabajo, y establece un punto de partida claro para las líneas de trabajo futuro.

7.7.1 Evaluación por nivel

La Tabla 17 evalúa el nivel TRL de esta investigación nivel por nivel, indicando el criterio general de cada nivel, la evidencia concreta que lo sustenta o lo justifica como pendiente, y el estado resultante:

TRL	Nombre	Criterio general	Evidencia en esta investigación	Estado
1	Principios básicos observados	Existe el principio científico que sustenta la tecnología.	La viabilidad de TinyML en microcontroladores y la integración IoT con NGSI están establecidas en la literatura	Superado

TRL	Nombre	Criterio general	Evidencia en esta investigación	Estado
			(Warden & Situnayake, 2019; FIWARE Foundation, 2020).	
2	Concepto de aplicación formulado	Se define cómo el principio científico puede aplicarse a un problema concreto.	La brecha entre TinyML e IoT estandarizado está identificada y formalizada en el Cap. 3. El concepto del agente inteligente embebido está definido en el Cap. 4 mediante la función f: P×H→A.	Superado
3	Prueba de concepto experimental	Primeras pruebas en laboratorio que demuestran la viabilidad del concepto.	La Fase 1 (Cap. 6) demostró que la plataforma genera agentes funcionales, los despliega en hardware real y completa el ciclo percepción–inferencia–transmisión FIWARE con el dataset de ocupación de habitaciones.	Superado

TRL	Nombre	Criterio general	Evidencia en esta investigación	Estado
4	Tecnología validada en laboratorio	La tecnología funciona en condiciones de laboratorio controladas con métricas cuantificables.	Los modelos TinyML (RF, NN, DT) se entrenaron, comprimieron y desplegaron en cuatro placas Arduino. Las métricas de latencia, RAM, Flash y tasa de transmisión fueron medidas y documentadas en la Fase 3.	Superado
5	Tecnología validada en entorno relevante	La tecnología se prueba en condiciones que simulan o representan el entorno real de uso.	El agente fue validado en dos escenarios de aplicación real: detección de caídas (exactitud 92 %, hardware real Nano 33 BLE + IMU LSM9DS1) y estimación de ETo (R^2=0.925, hardware real + DHT22 + anemómetro). Datos transmitidos a FIWARE en tiempo real.	Alcanzado

TRL	Nombre	Criterio general	Evidencia en esta investigación	Estado
6	Tecnología demostrada en entorno relevante	Prototipo funcional demostrado en condiciones representativas del entorno real.	La plataforma está en contenedores con Docker y desplegable localmente. Se realizaron pruebas de confiabilidad de 24 horas (R≥0.9994) y pruebas de transmisión en tiempo real con scripts reales a Orion+QuantumLeap+Grafana. La versión web pública está en desarrollo.	Parcial
7	Demostración en entorno operacional	Prototipo cercano al sistema final, probado en condiciones operacionales reales.	Requiere despliegue institucional con múltiples agentes simultáneos, validación por usuarios reales (adultos mayores, agricultores) y operación continua por semanas o meses. No realizado en esta investigación.	Pendiente

TRL	Nombre	Criterio general	Evidencia en esta investigación	Estado
8	Sistema completo y calificado	Sistema completo, probado y listo para producción.	Requiere certificación, pruebas de seguridad, documentación de usuario final y validación independiente por terceros.	Pendiente
9	Sistema en operación real	Tecnología desplegada y operando en condiciones reales de producción.	Requiere adopción institucional, mantenimiento continuo y base de usuarios activos.	Pendiente

Tabla 17. Evaluación del nivel de madurez tecnológica (TRL) de la plataforma propuesta

7.7.2 Nivel TRL alcanzado y justificación

Con base en el análisis de la Tabla 17, esta investigación alcanza un nivel TRL 5 consolidado y avanza parcialmente hacia TRL 6. Esta evaluación se fundamenta en los siguientes argumentos:

- TRL 1–4 superados con evidencia sólida. El fundamento teórico, la formalización matemática, la validación en laboratorio y las pruebas de compatibilidad de hardware están completamente documentados en los Capítulos 3 a 6 con métricas cuantitativas verificables.
- TRL 5 alcanzado mediante dos casos de estudio reales contrastantes. La validación en escenarios de salud (detección de caídas) y agricultura (estimación de ETo) con hardware real, datos reales y transmisión en tiempo real a FIWARE cumple el criterio de TRL 5: tecnología validada en un entorno relevante.
- TRL 6 parcialmente alcanzado. La containerización Docker, las pruebas de confiabilidad de 24 horas y los scripts de prueba en tiempo real demuestran un prototipo funcional representativo. Sin embargo, la versión web pública aún está en desarrollo y

no se realizó validación con usuarios finales reales en su entorno habitual, lo que impide declarar TRL 6 completo.

- TRL 7–9 fuera del alcance de esta investigación. Alcanzar estos niveles requiere despliegue institucional, validación por usuarios reales durante semanas o meses, certificación de seguridad y adopción en producción actividades que corresponden a la fase de transferencia tecnológica posterior a la investigación doctoral.

Comparación con el estado del arte: la mayoría de los trabajos revisados en el Capítulo 3 se ubican entre TRL 3 y TRL 4 demuestran inferencia embebida o integración IoT en condiciones de laboratorio, pero sin validación en escenarios reales ni pruebas de confiabilidad operativa. Esta investigación avanza al menos un nivel adicional al integrar ambas dimensiones y validarlas en condiciones reales.

7.7.3 Camino hacia TRL 7 y superiores

La Tabla 18 describe las acciones concretas necesarias para avanzar desde el nivel actual hacia TRL 7 y superiores, relacionándolas con las líneas de trabajo futuro propuestas en el Capítulo 8:

De TRL a TRL	Acciones requeridas	Relación con el trabajo futuro propuesto
6 → 7	Despliegue con usuarios reales en su entorno habitual. Validación con adultos mayores en domicilio y con agricultores en invernadero. Operación continua por al menos 4 semanas.	Incorporación de aprendizaje incremental para adaptar el modelo al comportamiento real del usuario. Evaluación de usabilidad de la plataforma con usuarios no expertos.
7 → 8	Certificación de seguridad del sistema, pruebas de estrés con decenas de agentes simultáneos, documentación de usuario final y auditoría independiente del código.	Extensión a sistemas multiagente y validación formal de la escalabilidad M/M/1 bajo carga real.
8 → 9	Adopción institucional (hospitales, dependencias	Colaboración con instituciones de salud, centros de

	agrícolas, plantas industriales), mantenimiento continuo y soporte técnico.	investigación agrícola e industria para despliegues piloto a escala.

Tabla 18. Acciones requeridas para avanzar hacia niveles TRL superiores y su relación con el trabajo futuro propuesto.

El análisis TRL confirma que esta investigación cumple el estándar esperado para una contribución doctoral en ingeniería de sistemas: alcanzar TRL 5 con evidencia de avance hacia TRL 6 es consistente con el objetivo de demostrar viabilidad tecnológica y sentar las bases para una transferencia tecnológica posterior. Los niveles TRL 7 a 9 corresponden a proyectos de aplicación industrial o médica que van más allá del alcance de una investigación básica, y su consecución requeriría colaboración institucional, financiamiento adicional y un horizonte temporal más amplio.

La discusión presentada en este capítulo permite formular cuatro conclusiones analíticas que trascienden los números del Capítulo 6:

- La compresión es el costo de la democratización. Los modelos TinyML embebidos tienen un rendimiento predictivo ligeramente inferior a sus versiones sin restricción de hardware. Esa diferencia de 2 puntos porcentuales en exactitud o 0.006 en R^2 es el precio de hacer viable la inteligencia artificial en hardware accesible de campo. Para las aplicaciones evaluadas, ese precio es aceptable.
- La comparación justa requiere contexto. El método de umbrales supera al Random Forest reducido en este dataset porque el dataset fue construido en condiciones de laboratorio que favorecen a las reglas simples. En condiciones reales con mayor variabilidad de usuario, el agente TinyML ofrece una ventaja estructural al poder actualizarse con nuevos datos.
- Las diferencias de rendimiento entre placas son de hardware, no de diseño. Al fijar el mismo modelo en todas las pruebas, se aisló el efecto del hardware. La portabilidad del agente quedó demostrada: el mismo código funciona en cuatro arquitecturas distintas sin modificaciones en la lógica del sistema.
- La aportación principal es la integración, no la precisión. El estado del arte muestra que TinyML e IoT han avanzado por separado. Esta tesis demuestra que pueden convergir en un agente embebido autónomo, generado automáticamente, estandarizado con NGSI v2 y conectado a plataformas industriales como FIWARE. Esa convergencia es la contribución diferenciadora.

El Capítulo 8 retoma estas conclusiones analíticas para formular los aportes finales de la investigación, las limitaciones que deben considerarse al generalizar los resultados y las líneas de trabajo futuro que emergen de este estudio.

Capítulo 8 – Conclusiones y Trabajo Futuro

8.1 Conclusiones generales

Esta tesis abordó el problema de la limitada capacidad de los dispositivos embebidos de bajo costo, como Arduino, para ejecutar procesos de inteligencia artificial e integrarse de manera efectiva en arquitecturas IoT modernas. A lo largo del trabajo se diseñó, implementó y evaluó una plataforma para la generación automática de agentes inteligentes embebidos, capaces de operar de forma autónoma en entornos reales y comunicarse mediante estándares IoT.

Los resultados experimentales demuestran que es posible ejecutar inferencia de modelos de aprendizaje automático directamente en microcontroladores con recursos restringidos, manteniendo latencias aceptables y un uso eficiente de memoria. Asimismo, se validó que estos agentes pueden integrarse de forma nativa con plataformas IoT estandarizadas como FIWARE, eliminando la necesidad de gateways intermedios y reduciendo la complejidad de despliegue.

La evaluación en dos dominios distintos salud y agricultura confirma que la arquitectura propuesta no está limitada a un caso de uso específico, sino que constituye un enfoque general para el diseño de agentes inteligentes embebidos. En conjunto, los resultados obtenidos respaldan ambas hipótesis de investigación: se demostró que es posible crear un agente inteligente de IoT embebido en Arduino con inferencia local, estandarización NGSI v2 y transmisión a FIWARE, y que es posible automatizar su generación mediante una plataforma software sin programación manual del firmware.

Es importante señalar que el modelo embebido en esta investigación opera de forma estática: aprende de datos históricos pero no actualiza sus parámetros en campo. Esto representa la limitación principal del trabajo y la línea de investigación más directa hacia la que apuntan las extensiones futuras, en particular mediante la incorporación de aprendizaje incremental en el dispositivo.

8.2 Cumplimiento de objetivos e hipótesis

El objetivo general de desarrollar una plataforma para la generación de agentes inteligentes en dispositivos Arduino integrados al Internet de las Cosas fue alcanzado satisfactoriamente. La plataforma permite configurar, generar y desplegar agentes capaces de analizar su entorno mediante modelos de inteligencia artificial embebida y comunicar los resultados a plataformas IoT estandarizadas.

Los objetivos específicos se cumplieron de la siguiente manera:

El objetivo 1 se cumplió mediante el análisis experimental del desempeño de distintos microcontroladores Arduino, evaluando sus capacidades y limitaciones para ejecutar algoritmos de IA.

El objetivo 2 se abordó bajo el enfoque de entrenamiento offline con datos generados en tiempo real, permitiendo inferencia embebida en tiempo real dentro de las restricciones del hardware.

El objetivo 3 se cumplió mediante el diseño e implementación de un modelo de comunicación eficiente basado en estándares IoT y su validación experimental.

El objetivo 4 se validó a través del entrenamiento de modelos con datos históricos y su adaptación para ejecución embebida en dispositivos Arduino.

El objetivo 5 se verificó mediante la evaluación del desempeño del agente considerando métricas de precisión, latencia, uso de memoria, confiabilidad y escalabilidad.

En consecuencia, ambas hipótesis quedan validadas con base en la evidencia experimental presentada. La Hipótesis 1 ¿es posible crear un agente inteligente de IoT capaz de ejecutar modelos de IA de forma local, estandarizar datos en NGSI v2 y transmitirlos a FIWARE desde un Arduino? queda validada por los resultados del Capítulo 6: latencias de 25–160 ms, confiabilidad operativa $R \geq 0.9994$, y transmisión exitosa al stack FIWARE en los cuatro dispositivos evaluados.

La Hipótesis 2 ¿es posible automatizar el proceso de creación del agente sin programación manual? queda validada por la plataforma R/Shiny que genera firmware funcional desplegable en un proceso de cinco pasos sin intervención técnica del usuario en el código del agente.

8.3 Contribuciones principales de la tesis

Las contribuciones de esta tesis se agrupan en científicas, tecnológicas y metodológicas, lo que refleja el carácter integral del trabajo.

Contribuciones científicas

- Se propone y valida un marco conceptual de agente inteligente embebido, donde la inteligencia artificial, la adquisición de datos y la comunicación IoT se integran en una sola entidad que opera directamente en el microcontrolador.
- Se aporta evidencia experimental que demuestra la viabilidad del edge intelligence real, trasladando la inferencia desde la nube o el gateway hacia dispositivos de muy bajo costo.
- Se contribuye al campo de TinyML al mostrar que la IA ligera puede formar parte de sistemas IoT completos y operativos, y no solo evaluarse como modelos aislados.

Contribuciones tecnológicas

- Se desarrolló una plataforma automatizada para la generación de agentes inteligentes, capaz de producir código embebido que integra modelos de IA, funciones de preprocesamiento y comunicación IoT estandarizada.
- Se eliminó la dependencia de gateways intermedios, simplificando la arquitectura IoT y reduciendo costos de despliegue.
- Se logró la integración nativa con FIWARE mediante el uso del estándar NGSI v2, garantizando interoperabilidad y reutilización de datos.
- Se validó la portabilidad del agente en múltiples placas Arduino, proporcionando una guía práctica para la selección de hardware según el caso de uso.

Contribuciones metodológicas

- Se definió una metodología experimental replicable para evaluar agentes inteligentes embebidos, considerando no solo métricas de IA, sino también latencia, uso de memoria, confiabilidad y escalabilidad.
- Se estableció una trazabilidad clara entre objetivos, métricas y resultados experimentales, facilitando la evaluación integral del agente como sistema.
- Se propuso un enfoque de evaluación que integra pruebas en entorno de desarrollo y en hardware real, fortaleciendo la validez de los resultados.

8.4 Trabajo futuro

8.4.1 Alcances y limitaciones del proyecto

El alcance de esta investigación doctoral abarca el diseño, implementación y validación experimental de una plataforma para la generación automática de agentes inteligentes embebidos en dispositivos Arduino integrados a ecosistemas IoT. Los alcances concretos son: (1) soporte para tres modelos de aprendizaje automático (Random Forest, Red Neuronal cuantizada y Árbol de Decisión), cuatro placas Arduino (Uno, Nano 33 BLE, ESP32 y Portenta H7 Lite) y tres protocolos de comunicación (HTTP, MQTT y CoAP); (2) integración nativa con FIWARE mediante el estándar NGSI v2 y el stack Orion–QuantumLeap–Grafana; y (3) validación experimental en dos dominios de aplicación reales y contrastantes salud y agricultura, con pruebas de confiabilidad operativa de 24 horas continuas y análisis de escalabilidad mediante teoría de colas.

Las limitaciones que deben considerarse al generalizar los resultados son las siguientes. Primera, el entrenamiento se realiza de forma offline: el agente embebido ejecuta inferencia con un modelo estático entrenado previamente y no aprende de las nuevas observaciones que recibe en campo. Si el fenómeno que monitorea cambia con el tiempo los patrones de movimiento de un adulto mayor o las condiciones climáticas de una región, el rendimiento del modelo puede degradarse gradualmente sin que el agente pueda adaptarse de forma autónoma.

Segunda, la validación experimental se realizó en condiciones de laboratorio y entornos representativos, pero no con usuarios finales en su entorno habitual durante períodos prolongados, por lo que el sistema alcanza TRL 5 con avance parcial hacia TRL 6 pero no niveles superiores.

Tercera, la plataforma fue evaluada con datasets estructurados de tamaño moderado; su comportamiento con flujos de datos de muy alta frecuencia (>500 Hz) o con sensores no convencionales no ha sido caracterizado. Cuarta, las pruebas de escalabilidad se basaron en modelos analíticos M/M/1 verificados con dos agentes simultáneos; la validación empírica con decenas de agentes en producción real queda fuera del alcance de este trabajo. Estas limitaciones no invalidan las contribuciones, sino que las sitúan honestamente dentro del estándar esperado para una investigación doctoral en ingeniería.

8.4.2 Trabajo futuro

Las limitaciones identificadas y las oportunidades detectadas durante la investigación dan lugar a tres líneas de trabajo futuro organizadas por horizonte temporal.

Corto plazo — Aprendizaje incremental en el agente embebido

La limitación más directamente tratable es la naturaleza estática del modelo embebido. El aprendizaje incremental también denominado online learning o continual learning es la capacidad de un modelo de actualizar sus parámetros con nuevas observaciones de forma

continua, sin necesidad de reentrenar desde cero con el dataset completo original. A diferencia del aprendizaje offline empleado en esta tesis, donde el modelo se “congela” antes de ser embebido, un agente con aprendizaje incremental ajusta sus parámetros internamente con cada nuevo ciclo de percepción. Esto le permite adaptarse a la deriva de concepto (concept drift) el fenómeno por el cual la distribución estadística de los datos cambia gradualmente con el tiempo. En el caso de detección de caídas, el agente podría adaptarse a los cambios en los patrones de movimiento de un usuario específico a medida que envejece o se recupera de una lesión. En el caso de estimación de ETo, podría recalibrar su modelo ante cambios climáticos estacionales sin intervención manual.

El principal desafío técnico del aprendizaje incremental en Arduino es el olvido catastrófico: la tendencia de los modelos de redes neuronales a olvidar el conocimiento previo cuando se actualizan con nuevos datos. Para los modelos de árbol empleados en esta tesis existen propuestas como los Árboles de Hoeffding (Very Fast Decision Trees), que permiten aprendizaje incremental nativo con garantías estadísticas sobre cuándo una rama debe actualizarse. Trabajos como TyBox [25] han demostrado viabilidad de aprendizaje incremental en Arduino Nano 33 BLE para clasificación de imágenes y reconocimiento de actividad. La incorporación de esta capacidad al agente de esta tesis requeriría modificar la sección 4.5.2 de implementación del historial H, para que además de construir la entidad NGSI, el agente actualice parcialmente los parámetros del modelo con la observación reciente, almacenando los parámetros actualizados en la EEPROM del dispositivo para persistencia entre ciclos de energía.

Mediano plazo — Arquitecturas multiagente y validación de escalabilidad

En el mediano plazo, se plantea la extensión del sistema hacia arquitecturas multiagente distribuidas, donde múltiples dispositivos Arduino desplegados en un mismo entorno coordinan sus inferencias y decisiones. En esta arquitectura, un agente líder (leader agent) agregaría las predicciones de múltiples agentes especializados, aplicaría lógica de votación o fusión de datos, y transmitiría una predicción consolidada al broker FIWARE. Esta extensión permitiría además validar empíricamente el modelo de escalabilidad M/M/1 planteado en la sección 4.3 con decenas de agentes simultáneos reales, superando la limitación analítica de esta tesis.

Largo plazo — Validación con usuarios reales y transferencia tecnológica

En el largo plazo, se propone la validación de la plataforma con usuarios reales en sus entornos habituales durante períodos prolongados, avanzando de TRL 5 hacia TRL 7 y superiores. Para el caso de detección de caídas, esto implica desplegar el agente con adultos mayores en domicilio durante al menos cuatro semanas, midiendo tasa de detección real, falsas alarmas, aceptación del dispositivo y usabilidad de la plataforma para personal de salud no experto en programación. Para el caso de estimación de ETo, implica validación con agricultores en invernadero durante una temporada completa, comparando el ahorro de agua y el rendimiento de los cultivos respecto a métodos tradicionales. Estos estudios de campo sentarían las bases para la transferencia tecnológica institucional en colaboración con dependencias de salud pública y centros de investigación agrícola.

Apéndice A – Guía de Replicación

Este apéndice proporciona instrucciones detalladas para que un investigador externo pueda instalar, configurar y utilizar la plataforma desarrollada en esta tesis, así como replicar los experimentos presentados en el Capítulo 6.

1. Requisitos previos

Antes de instalar la plataforma, la máquina debe contar con:

- Docker Desktop
- Docker Compose plugin
- PowerShell
- conexión a internet

Opcionalmente:

- Git for Windows
- Visual Studio Code

Verificación:

```
docker --version
docker compose version
```

2. Obtener el proyecto

Opción 1: copiar carpeta

Copiar la carpeta completa del proyecto a la nueva máquina.

Opción 2: clonar desde Git

```
git clone <URL_DEL_REPOSITORIO>
cd Python-3.10.19
```

3. Crear el archivo de entorno

Desde la raíz del proyecto:

```
Copy-Item .env.platform.example .env.platform
```

4. Ajustar configuración del entorno

Abre `.env.platform` y revisa estos valores:

```
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
POSTGRES_DB=iot
POSTGRES_HOST_PORT=5433

BACKEND_PORT=8000
FRONTEND_PORT=3000

DATABASE_URL=postgresql+psycopg2://postgres:postgres@db:5432/iot
SECRET_KEY=change-this-in-real-installation
ACCESS_TOKEN_EXPIRE_MINUTES=60
MAX_UPLOAD_MB=50

REACT_APP_API_URL=http://localhost:8000
ORION_URL=http://orion:1026/v2/entities

MONGO_HOST_PORT=27017
ORION_HOST_PORT=1026
CRATE_HTTP_PORT=4200
CRATE_PSQL_PORT=5434
QUANTUMLEAP_PORT=8668
GRAFANA_PORT=3001
```

5. Instalar solo la plataforma

Si solo quieres backend, frontend y base de datos:

```
docker compose -f docker-compose.platform.yml up -d --build
```

Verificar:

```
docker compose -f docker-compose.platform.yml ps
```

Accesos:

- Frontend: `http://localhost:3000`
- Backend: `http://localhost:8000`

6. Instalar plataforma + FIWARE

Si quieres levantar también Orion, MongoDB, QuantumLeap, CrateDB y Grafana:

```
docker compose -f docker-compose.platform.yml --profile fiware up -d --build
```

Verificar:

```
docker compose -f docker-compose.platform.yml --profile fiware ps
```

Accesos:

- Plataforma: `http://localhost:3000`
- API: `http://localhost:8000`
- Orion: `http://localhost:1026`
- QuantumLeap: `http://localhost:8668`
- Grafana: `http://localhost:3001`

7. Validar backend

```
curl.exe -i http://localhost:8000/
```

8. Validar Orion

```
curl.exe -i http://localhost:1026/version
```

9. Revisar logs si algo falla

Backend:

```
docker compose -f docker-compose.platform.yml logs backend --tail 200
```

Frontend:

```
docker compose -f docker-compose.platform.yml logs frontend --tail 200
```

Orion:

```
docker compose -f docker-compose.platform.yml --profile fiware logs orion --tail 200
```

Mongo:

```
docker compose -f docker-compose.platform.yml --profile fiware logs mongo-db --tail 200
```

QuantumLeap:

```
docker compose -f docker-compose.platform.yml --profile fiware logs quantumleap --tail 200
```

10. Crear usuario e iniciar sesión

Abre:

```
http://localhost:3000
```

Luego:

- registra un usuario
- inicia sesión

11. Subir un dataset

En la interfaz:

- entra a `Upload / View Datasets`
- sube un dataset CSV
- verifica preview y estadísticas

12. Entrenar un modelo

En `Train Models`:

- selecciona dataset
- elige tipo de tarea
- elige algoritmo
- define target y features
- ejecuta entrenamiento

Después revisa el resultado en `Validate Models`.

13. Probar datos en tiempo real directos a FIWARE

Detección de caídas

```
python scripts\send_fiware_realtime.py `
  --orion-url http://localhost:1026/v2/entities `
  --service openiot `
  --service-path / `
  --scenario fall-detection `
  --entity-id fall_sensor_001 `
  --count 120 `
  --interval 0.5 `
  --create-subscription `
  --notify-url http://quantumleap:8668/v2/notify
```

Evapotranspiración

```
python scripts\send_fiware_realtime.py `
  --orion-url http://localhost:1026/v2/entities `
  --service openiot `
  --service-path / `
  --scenario eto `
  --entity-id eto_station_001 `
  --count 120 `
  --interval 1 `
  --create-subscription `
  --notify-url http://quantumleap:8668/v2/notify
```

14. Abrir Grafana

`http://localhost:3001`

Credenciales típicas iniciales:

- usuario: `admin`
- contraseña: `admin`

15. Consultar datos históricos en Grafana

Caídas

```
SELECT
  "time_index" AS "time",
  accel_x,
  accel_y,
  accel_z,
  fall_detected
FROM mtopeniot.etdevice
WHERE entity_id = 'fall_sensor_001'
  AND $__timeFilter("time_index")
ORDER BY "time_index"
```

Evapotranspiración

```
SELECT
  "time_index" AS "time",
  temperature_c,
  relative_humidity,
  wind_speed_m_s,
  eto_mm_day
FROM mtopeniot.etdevice
WHERE entity_id = 'eto_station_001'
  AND $__timeFilter("time_index")
ORDER BY "time_index"
```

16. Detener los servicios

Plataforma sola

```
docker compose -f docker-compose.platform.yml down
```

Plataforma + FIWARE

```
docker compose -f docker-compose.platform.yml --profile fiware down
```

17. Reanudar servicios

Plataforma sola

```
docker compose -f docker-compose.platform.yml up -d
```

Plataforma + FIWARE

```
docker compose -f docker-compose.platform.yml --profile fiware up -d
```

Referencias

[1] D. Evans, “The Internet of Things: How the Next Evolution of the Internet Is Changing Everything,” Cisco IBSG, 2011.

[2] K. Ashton, “That ‘Internet of Things’ Thing,” RFID Journal, 2009.

[3] L. Atzori, A. Iera, y G. Morabito, “The Internet of Things: A survey,” Computer Networks, vol. 54, no. 15, pp. 2787–2805, 2010.

[4] FIWARE Foundation, “Orion Context Broker Manual,” 2020.

[5] Amazon Web Services, “AWS IoT Core Documentation,” 2021.

[6] H. Karimipour y V. Dinavahi, “On False Data Injection Attack Against Dynamic State Estimation on Smart Power Grids,” IEEE Transactions on Smart Grid, 2018.

[7] A. F. Skarmeta, “NGSI Data Models for IoT Interoperability,” FIWARE White Paper, 2020.

[8] M. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed., Wiley, 2009.

[9] P. Warden y D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O’Reilly Media, 2019.

[10] S. Banbury et al., “Benchmarking TinyML Systems,” arXiv preprint arXiv:2003.04821, 2020.

[11] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[12] S. Russell y P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed., Pearson, 2021.

[13] T. Hastie, R. Tibshirani y J. Friedman, *The Elements of Statistical Learning*, Springer, 2009.

[14] J. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[15] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, 4th ed., McGraw-Hill, 2002.

[16] L. Kleinrock, *Queueing Systems, Volume I: Theory*. Wiley, 1975.

[17] M. J. Page et al., “The PRISMA 2020 statement: an updated guideline for reporting systematic reviews,” *BMJ*, vol. 372, n71, 2021. doi: 10.1136/bmj.n71

[18] R. G. Allen, L. S. Pereira, D. Raes, y M. Smith, *Crop Evapotranspiration: Guidelines for Computing Crop Water Requirements*, FAO Irrigation and Drainage Paper No. 56. Food and Agriculture Organization of the United Nations, Rome, 1998.

[19] A. Bourke et al., “Fall detection — Principles and Methods,” in *Proc. 29th Annual International Conference of the IEEE EMBS*, Lyon, France, 2007, pp. 1663–1666.

[20] European Commission, “Technology Readiness Levels (TRL),” *Horizon 2020 — Work Programme 2014–2015: General Annexes*, Brussels, 2014.

[30] S. Franklin y A. Graesser, “Is it an Agent, or Just a Program? A Taxonomy for Autonomous Agents,” en *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer, Budapest, Hungary, pp. 21–35, 1996.

[31] N. R. Jennings, “On Agent-Based Software Engineering,” *Artificial Intelligence*, vol. 117, no. 2, pp. 277–296, 2000. doi: 10.1016/S0004-3702(99)00107-1

[21] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki y A. S. Hafid, “A Comprehensive Survey on TinyML,” *IEEE Access*, vol. 11, pp. 96892–96922, 2023. doi: 10.1109/ACCESS.2023.3294111

[22] R. Immonen y T. Hämäläinen, “Tiny Machine Learning for Resource-Constrained Microcontrollers,” *Journal of Sensors*, vol. 2022, 7437023, 2022. doi: 10.1155/2022/7437023

[23] S. Hymel et al., “Edge Impulse: An MLOps Platform for Tiny Machine Learning,” *arXiv preprint* arXiv:2212.03332, 2022.

[24] B. Sudharsan et al., “OTA-TinyML: Over the Air Deployment of TinyML Models and Execution on IoT Devices,” *IEEE Internet Computing*, vol. 26, no. 3, pp. 69–78, 2022. doi: 10.1109/mic.2021.3133552

[25] M. Pavan, E. Ostrovan, A. Caltabiano y M. Roveri, “TyBox: An automatic design and code generation toolbox for TinyML incremental on-device learning,” *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 3, pp. 1–27, 2024. doi: 10.1145/36045

[26] X. Yu et al., “A practical wearable fall detection system based on tiny convolutional neural networks,” *Biomedical Signal Processing and Control*, vol. 86, 105290, 2023. doi: 10.1016/j.bspc.2023.105290

[27] J. Hu, F. Cheng, M. Liu, X. Xu y X. Li, “MicroFallNet: A lightweight model for real-time fall detection on smart wristbands,” *Pervasive and Mobile Computing*, vol. 109, 102046, 2025. doi: 10.1016/j.pmcj.2025.102046

[28] Z. Hu et al., “Machine Learning Based Prediction of Reference Evapotranspiration (ET0) Using IoT,” *IEEE Access*, vol. 10, pp. 70526–70540, 2022. doi: 10.1109/access.2022.3187528

[29] R. N. Bashir et al., “Smart reference evapotranspiration using Internet of Things and hybrid ensemble machine learning approach,” *Internet of Things*, vol. 24, 100962, 2023. doi: 10.1016/j.iot.2023.100962