UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Reconocimiento de Objetos Usando TinyML en Dispositivos IoT

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: García Antón, Juan

Tutor/a: Manzoni, Pietro

CURSO ACADÉMICO: 2024/2025

Resumen

El presente Trabajo de Fin de Grado se orienta al desarrollo de un prototipo capaz de verificar, en tiempo real, el uso de casco de seguridad en el acceso a zonas consideradas de riesgo dentro de un entorno industrial. El punto de partida fue la necesidad de disponer de un sistema de control que no dependiera únicamente de inspecciones manuales ni de plataformas centralizadas con alta complejidad de despliegue. La propuesta se basó en comprobar si un dispositivo embebido podía asumir todo el proceso de identificación, comprobación visual y registro de los intentos de acceso.

El sistema se construyó sobre la placa Portenta H7 junto con la Vision Shield, que incorpora una cámara. A este conjunto se añadió un lector RFID encargado de asociar cada tarjeta a un operario concreto, así como un almacenamiento en microSD para garantizar la trazabilidad de los registros. Cuando existía conectividad, los datos se sincronizaban con una base de datos en la nube, manteniendo de este modo un doble nivel de persistencia. De esta forma, el flujo completo quedaba integrado en un único dispositivo.

El proyecto no se planteó como una solución definitiva, sino como una validación técnica de viabilidad. Los resultados obtenidos muestran que es posible ejecutar la detección del casco mediante un modelo de visión artificial embebida con tiempos de respuesta adecuados, y mantener un registro seguro incluso en ausencia de conexión. Estos hallazgos abren la puerta a futuras ampliaciones, como la inclusión de otros equipos de protección individual o la integración con mecanismos reales de control de acceso.

Palabras clave: IoT, visión artificial, TinyML, Edge Impulse, EPIs, Portenta H7, RFID, Supabase.

Abstract

This Bachelor's Thesis focuses on the development of a prototype capable of verifying, in real time, the use of a safety helmet when accessing designated risk areas within an industrial environment. The starting point was the need for a control system that would not rely solely on manual inspections or on centralized platforms with high deployment complexity. The aim was to evaluate whether an embedded device could handle the entire process of identification, visual verification, and event logging.

The system was built on the Portenta H7 board together with the Vision Shield, which integrates a camera. An RFID reader was added to associate each card with a specific worker, as well as microSD storage to ensure traceability of records. When connectivity was available, data were synchronized with a cloud database, thus maintaining a dual level of persistence. In this way, the complete workflow remained integrated within a single device.

The project was not conceived as a final market-ready solution, but rather as a technical feasibility validation. The results demonstrate that helmet detection through an embedded computer vision model can be achieved with suitable response times, while ensuring reliable logging even in the absence of network connectivity. These findings open the door to future extensions, such as incorporating other types of personal protective equipment or integrating the system with real access control mechanisms.

Keywords: IoT, computer vision, TinyML, Edge Impulse, PPE, Portenta H7, RFID, Supabase.

Índice

Índice iv
Lista de figuras vi

Capítulo 1: Introducción 1
1.1. Motivación 1
1.2. Objetivos del proyecto 2
1.3. Metodología general 2
1.4. Estructura de la memoria 3
Capítulo 2: Estado del arte 5
2.1. Edge Computing y TinyML 5
2.2. Detección de EPIs en la industria 6
2.3. Modelos de visión artificial embebida 7
2.4. Tecnologías y dispositivos hardware 9
2.4.1. Portenta H7 9
2.4.2. Vision Shield LoRa 10
2.4.3. Lector RFID 11
Capítulo 3: Diseño del sistema 14
3.1. Diagrama general del sistema 14
3.2. Flujo de funcionamiento del sistema 18
3.3. Requisitos funcionales y no funcionales 19
3.3.1 Requisitos funcionales 19
3.3.2 Requisitos no funcionales 19
3.4. Limitaciones hardware 21
Capítulo 4: Implementación 22
4.1. Integración de componentes 22
4.2. Captura del dataset 23
4.3. Entrenamiento del modelo en EdgeImpulse 26
4.3.1 Configuración inicial 26
4.3.2 Configuración final 27
4.4. Programación de la lógica del sistema en la Portenta H7 28
4.4.1 main.py 28

4.4.2 storage_local.py ... 28
4.4.3 cloud_sync.py ... 29
4.4.4 cards_sync.py ... 29
4.4.5 wifi_setup.py ... 29
4.4.6 Compatibilidad con SSL y peticiones HTTP ... 30
Capítulo 5: Evaluación experimental y resultados ... 31
5.1. Resultados de validación y test ... 31
5.2. Rendimiento en dispositivo ... 32
5.3. Evaluación del sistema completo ... 33
5.4. Evaluación del flujo de acceso ... 33
5.5. Comportamiento sin conexión Wi-Fi ... 34
5.6. Robustez y limitaciones observadas ... 34
5.7. Casos límite y observación de fallos ... 35
5.8. Indicadores de rendimiento ... 35
5.9. Reflexiones finales sobre la evaluación ... 36
Capítulo 6: Conclusiones y trabajos futuros ... 38
6.1 Conclusiones generales ... 38
6.2 Aportaciones principales ... 39
6.3 Limitaciones detectadas ... 39
6.4 Trabajos futuros ... 40
6.4.1 Ampliación del alcance de detección ... 40
6.4.2 Mejora de la calidad visual ... 40
6.4.3 Optimización del hardware y del rendimiento ... 40
6.4.4 Integración con sistemas de acceso reales ... 41
6.4.5 Refuerzo de la seguridad y la fiabilidad ... 41
6.4.6 Evaluación en entorno reales ... 41
Bibliografía ... 42

Anexo A: Código fuente resumido ... 44
Anexo B: Contribución a los Objetivos de Desarrollo Sostenible ... 45

Lista de figuras

Figura 1. Tabla de objetivos del proyecto. 2
Figura 2. Secuencia de trabajo 3
Figura 3. Portenta H7 9
Figura 4. Vision Shield LoRa 10
Figura 5. Lector de tarjetas RFID 11
Figura 6. Tarjetas RFID 125 KHz 11
Figura 7. Diagrama general del sistema 14
Figura 8. Conexión de un pulsador al puerto GPIO D2 de la Portenta 15
Figura 9. Lector RFID, tarjetas y FA 12V. 15
Figura 10. Conexión Sistema independiente con FA, RFID, Portenta H7. 16
Figura 11. Flujo de funcionamiento del sistema 18
Figura 12. Tabla de relación objetivo-requisito. 20
Figura 13. Esquema de conexión 22
Figura 14. Ejemplo del primer dataset (defectuoso) 23
Figura 15. Foto de cómo se implementó el pulsador 24
Figura 16. Imágenes del diseño de la caja y los soportes para su impresa en 3D. 24
Figura 17. Imágenes del sistema de captura con la caja impresa en 3D. 24
Figura 18. Ejemplo del segundo dataset (ejemplo con casco). 25
Figura 19. Ejemplo del segundo dataset (ejemplo sin casco) 25
Figura 20. Configuración inicial en Edge Impulse 26
Figura 21. Configuración final en Edge Impulse 27
Figura 22. Gráfica de dispersión de características del modelo 31
Figura 23. Matríz de confusión del modelo 32
Figura 24. Resultados de rendimiento del modelo 32
Figura 25. Tabla de resultados globales del sistema. 35
Figura 26. Tabla de validación de requisitos del sistema. 37

Capítulo 1: Introducción

1.1. Motivación

En tercer curso de Ingeniería Informática, cursando la asignatura de Internet de las Cosas fue cuando realmente me interesó este tema. Ya había oído hablar de ello, pero nunca lo había trabajado en serio. Me llamó mucho la atención lo que se podía hacer juntando dispositivos pequeños con un poco de lógica. Me sorprendió que con tan poco, se pudiera montar algo que funcionara de forma autónoma, según los eventos que sucedan a su alrededor.

Esa asignatura fue bastante práctica, y eso me gustó. Tuvimos que conectar sensores, módulos, probar cosas… resolver situaciones concretas. Nada demasiado complejo, pero suficiente para incentivar ampliar mi curiosidad, fue ahí cuando decidí que esta podría ser la línea principal de mi TFG.

Cuando empecé el proyecto no buscaba hacer un simple experimento. Quería montar algo que pudiera funcionar de verdad, que tuviera una aplicación real y solucionase un problema. Me pareció que en el sector industrial encajaba bien. El tema de la seguridad es muy importante, así que pensé en un sistema que comprobase si un trabajador lleva los EPIs antes de entrar en zona de peligro, es una buena línea de trabajo.

Durante todo el proceso he tenido que aprender muchas cosas nuevas. He usado la Portenta H7, entrenado modelos con Edge Impulse, trabajado Edge functions, usado sensores RFID… Algunas partes me costaron bastante, sobre todo al principio. Hubo pruebas que fallaban, pero fui perseverante, probando -> corrigiendo -> validando, y poco a poco fue saliendo.

Con todo eso, he puesto en práctica muchas de las enseñanzas aprendidas a lo largo de la carrera. El sistema funciona, hace lo que se espera. No es perfecto, aún se puede pulir más, pero demuestra que la idea es viable y la solución es válida.

1.2. Objetivos del proyecto

El proyecto se planteó con una serie de objetivos concretos que permitieran guiar el desarrollo y orientar las decisiones técnicas en cada fase. Estos objetivos se definieron de forma que abarcaran tanto las funciones básicas del sistema como las condiciones necesarias para que el prototipo fuera viable en un entorno industrial.

O1	Detectar, en tiempo real, si un operario accede con el casco de seguridad correctamente colocado.
O2	Garantizar el funcionamiento del sistema en ausencia de conexión de red, conservando los registros de acceso de manera local.
O3	Asociar cada intento de acceso a la identidad de un trabajador mediante la lectura de tarjetas RFID.
O4	Ejecutar el modelo de visión artificial embebida con un nivel de precisión suficiente para distinguir entre las clases “casco” y “no casco”.
O5	Mantener la trazabilidad de los intentos de acceso mediante el registro local en la microSD y la posterior sincronización con la base de datos en la nube.
O6	Integrar todos los módulos desarrollados en un flujo coherente y estable, que pueda ejecutarse de manera continua en la Portenta H7.

Figura 1. Tabla de objetivos del proyecto.

1.3. Metodología general

La metodología no se concibió como un plan rígido desde el principio. Más bien se trabajó de manera progresiva, avanzando por módulos que podían evaluarse por separado. Se empezó con la captura de imágenes para el dataset y entrenamiento del modelo, después por probar la lectura de tarjetas RFID, junto a la inferencia local, y de forma independiente se abordó la parte de almacenamiento y comunicación. Cada bloque se probó con ejemplos simples, lo que permitía descartar fallos de base antes de combinarlos.

A lo largo del proceso se fueron registrando incidencias y pequeños ajustes que surgían, ya que algunos componentes no respondían a la primera. En varios momentos fue necesario repetir pruebas o cambiar parámetros hasta obtener un funcionamiento estable. Una vez alcanzada esa estabilidad mínima en cada módulo, se procedió a la integración, aunque no siempre de manera lineal: en ocasiones hubo que volver atrás y modificar lo ya hecho. El desarrollo avanzó de este modo, con iteraciones sucesivas que facilitaron introducir mejoras sin comprometer el conjunto.

Figura 2. Secuencia de trabajo

Los detalles concretos de la evolución por versiones se explican en el capítulo 3, donde se muestran también los principales cambios y ajustes realizados.

1.4. Estructura de la memoria

Este documento se ha organizado en seis capítulos, bibliografía y varios anexos complementarios con material técnico.

- Capítulo 1: Introducción - Aquí se presenta el punto de partida del trabajo. Incluye la motivación personal que llevó a elegir el tema, el objetivo principal del proyecto y la metodología general que se ha seguido durante el desarrollo. También se explica brevemente cómo se estructura el documento para orientar al lector.

- Capítulo 2: Estado del arte - En este capítulo se repasan las tecnologías y enfoques relacionados con el proyecto. Se empieza con una visión general sobre Edge Computing y TinyML, se analizan trabajos previos sobre detección de EPIs en la industria, se describen los modelos de visión artificial embebida y, por último, se comentan los dispositivos hardware que forman parte del sistema (Portenta H7, Vision Shield, lector RFID). Sirve para situar el proyecto en el contexto de lo que ya existe.

- Capítulo 3: Diseño del sistema - Aquí se describe la arquitectura planteada antes de la implementación. Se incluye un diagrama general del sistema, el flujo de funcionamiento paso a paso y la definición de los requisitos técnicos y funcionales. También se detallan las limitaciones de hardware que condicionan las decisiones de diseño. Es la parte que justifica cómo y por qué se han organizado los componentes de la forma elegida.

- Capítulo 4: Implementación - Este capítulo cuenta cómo se llevó a cabo el sistema en la práctica. Se explica la integración de componentes, la captura y preparación del dataset, el entrenamiento del modelo en Edge Impulse y la programación de la lógica final en la Portenta H7. También se incluye la parte de almacenamiento en microSD y la sincronización de eventos en Supabase mediante conexión Wi-Fi.

- Capítulo 5: Evaluación experimental y resultados - En este apartado se presentan las pruebas realizadas, tanto a nivel de métricas del modelo (precisión, recall, F1, matrices de confusión) como en el uso real sobre la Portenta. Se compara el rendimiento entre diferentes datasets y configuraciones, y se analizan los casos problemáticos detectados durante las pruebas. La idea es mostrar la fiabilidad del sistema y qué limitaciones se han encontrado.

- Capítulo 6: Conclusiones y trabajos futuros - Aquí se recogen las conclusiones principales del proyecto: qué se ha conseguido, qué límites se han encontrado y qué posibles mejoras o líneas de trabajo se podrían abordar en el futuro. Por ejemplo, ampliar la detección a otros EPIs, mejorar la cámara, optimizar la sincronización o incluso explorar de nuevo la comunicación con LoRaWAN [1] en entornos con infraestructura adecuada.

Al final del documento se añaden la bibliografía y, en caso necesario, un anexo con código, esquemas o capturas relevantes para complementar la información del proyecto.

Capítulo 2: Estado del arte

2.1. Edge Computing y TinyML

Hoy en día, no todo el procesamiento de datos necesita hacerse en la nube. En muchos casos, como ocurre en entornos industriales, resulta más práctico y eficiente que el propio dispositivo realice la inferencia directamente. A esto se le llama Edge Computing. En lugar de enviar una imagen a un servidor para analizarla, el dispositivo la procesa ahí mismo, justo cuando la capta. Esta forma de trabajar tiene muchas ventajas: por un lado, reduce el tiempo de respuesta; por otro, no depende de que haya una conexión de red activa. También mejora la privacidad, ya que los datos sensibles, como imágenes de personas, no son expuestas, al no salir nunca del sistema. Como se explica en la documentación de Edge Impulse [2], esta estrategia es clave en dispositivos de bajo consumo y en escenarios donde la fiabilidad o la seguridad de red no están garantizadas.

Relacionado con esto está el concepto de TinyML [3]. Básicamente, es una forma de aplicar aprendizaje automático en microcontroladores, es decir, en dispositivos con recursos muy limitados. Modelos que normalmente se ejecutarían en ordenadores se adaptan para que tengan cabida y funcionen correctamente en placas con poca memoria y baja potencia de cálculo. Se utilizan modelos más pequeños y optimizados, con un enfoque específico en el consumo energético y la eficiencia. La comunidad científica define TinyML como una combinación de técnicas de compresión, cuantización y despliegue inteligente para llevar inteligencia artificial a dispositivos embebidos, especialmente aquellos con menos de 1 MB de memoria RAM.

Este enfoque tiene mucho sentido cuando se pretende desplegar un sistema como el planteado en este proyecto, ya que no depende de conexiones permanentes ni de servidores externos. Si se quisiera hacer lo mismo desde la nube, habría que montar una infraestructura de red más robusta, almacenar datos personales y asegurarse de que todo funcione en tiempo real, lo cual no siempre es viable. Con Edge Computing y TinyML, en cambio, cada dispositivo puede operar por su cuenta, con independencia del resto. Si un nodo falla, el resto sigue funcionando.

Por supuesto, también hay limitaciones. No se puede cargar cualquier modelo, y a veces hay que hacer pruebas para ajustar el equilibrio entre precisión y eficiencia. Pero aun con esas limitaciones, este tipo de soluciones permiten construir sistemas funcionales y aplicables en situaciones reales, sin necesidad de equipos caros ni de instalaciones complejas. En este trabajo, se aprovechan precisamente esas ventajas para construir un sistema capaz de detectar equipos de protección individual directamente desde una placa embebida.

2.2. Detección de EPIs en la industria

Comprobar que los trabajadores llevan el EPI no es algo nuevo. Es de lo más básico en seguridad laboral y, en industrias como la construcción, la minería o la fabricación pesada, puede marcar la diferencia entre un día normal o un accidente serio. Y no solo es cuestión de salud. La normativa de seguridad industrial [4] es estricta: si no se cumplen, llegan sanciones, paradas de producción… y eso cuesta dinero. Por eso las empresas llevan años buscando maneras de vigilar de forma eficiente su cumplimiento, y lo han hecho de muchas formas, algunas bastante rudimentarias y otras mucho más avanzadas.

Lo más habitual, y lo que todavía se ve mucho, es la inspección visual manual. Un supervisor se pasea o controla desde una zona elevada y verifica que todo el mundo lleva los EPIs [5]. Funciona, pero tiene problemas claros: hace falta personal dedicado, la vista humana falla, y en plantas grandes o con mucha gente, es imposible estar en todas partes a la vez. Además, estar encima de la gente todo el rato no es precisamente cómodo para nadie.

Luego llegaron los sensores dedicados. Cosas como sistemas de acceso que solo abren si detectan un chip RFID en el casco o sensores de presión integrados en ciertos equipos. La idea es buena, pero aquí tampoco se ve realmente el EPI correctamente colocado: el sistema asume que está puesto porque detecta el chip o el sensor, pero no avisa si el EPI está mal colocado [6].

El siguiente paso fue poner cámaras de vigilancia. Al principio no hacían más que grabar y enviar la señal a una sala de control, donde otra persona tenía que mirar varias pantallas y confirmar que todo estaba en orden. El cambio es que ya no hacía falta estar físicamente en el sitio, pero el trabajo seguía siendo manual, con los mismos problemas de cansancio y atención.

Aquí es donde la cosa dio un salto con la visión por computador y el aprendizaje profundo. Ahora sí, las cámaras podían “ver” y reconocer por sí mismas si alguien llevaba casco. Esto se consiguió gracias a redes neuronales convolucionales y, más tarde, a modelos más ligeros pensados para funcionar directamente en dispositivos pequeños, sin tener que depender de un servidor [7].

Existen ya varias empresas que han llevado la detección automática de EPIs al terreno práctico. Visionify [8], por ejemplo, ofrece un sistema basado en visión artificial que se integra con cámaras existentes y permite comprobar en tiempo real si los trabajadores llevan casco, chaleco reflectante, gafas o guantes, generando alertas cuando se incumplen las normas de seguridad. Otra compañía muy activa en este campo es viAct [9], cuya plataforma se ha implantado en sectores como la construcción, la minería o la logística, detectando la falta de protección personal y emitiendo avisos instantáneos para reducir accidentes. En España destaca la empresa Seitech, que ha desarrollado la solución Calidus [10], validada en empresas como Circet España [11], donde se comprobó con éxito el uso de cascos y otros equipos de protección en trabajos de campo. Estos ejemplos muestran cómo la visión por computador ya se aplica de forma directa en entornos industriales para reforzar la seguridad y garantizar la trazabilidad de los operarios.

Comparados con los antiguos métodos, estos nuevos sistemas tienen ventajas claras: vigilancia 24/7, no dependen tanto de la atención de una persona, se pueden conectar al control de accesos y dejan registros que sirven para auditorías.

No todo es perfecto. Si la luz cambia mucho, si la persona está parcialmente tapada o si el casco es de un diseño distinto al que ha visto el modelo, el rendimiento puede bajar. Y si hay que cubrir muchas zonas a la vez, el coste de instalar y mantener todo el sistema es elevado.

Actualmente la tendencia a combinar sistemas de IA en el borde con mecanismos de identificación y registro, especialmente en control de accesos y seguridad laboral e un nuevo nicho de mercado por explotar. Este enfoque permite procesar la información de manera local, lo que mejora la rapidez de respuesta y protege la privacidad, mientras que el registro automático de eventos asegura la trazabilidad sin necesidad de supervisión constante. Es precisamente en esa línea donde se sitúa este proyecto. Un sistema capaz de verificar en tiempo real el uso de casco, sin depender de la red para funcionar, y que además almacena de forma estructurada cada intento de acceso.

2.3. Modelos de visión artificial embebida

La visión por computador ha sido tradicionalmente un campo reservado a ordenadores potentes y servidores con gran capacidad de cálculo. En la década de 2010, la mayor parte de las aplicaciones prácticas dependían de sistemas centralizados, con cámaras que enviaban las imágenes a un servidor donde se ejecutaban los modelos. Esto limitaba mucho su uso en entornos donde la conectividad era irregular o donde se necesitaban respuestas rápidas.

A partir de 2012, con la aparición de la Raspberry Pi model B [12], comenzaron a popularizarse dispositivos compactos y de bajo coste capaces de ejecutar bibliotecas como OpenCV, lo que facilitó experimentar con sistemas de visión por computador fuera de entornos de laboratorio. Aunque supusieron un avance en accesibilidad, estos sistemas seguían dependiendo de alimentación constante o baterías de gran tamaño debido al consumos energético, por lo que todavía no podían considerarse soluciones de ultra bajo consumo típicas de los microcontroladores actuales [13]. En los últimos años la situación ha cambiado con la llegada de los microcontroladores capaces de ejecutar modelos de aprendizaje automático gracias a librerías y optimizaciones específicas. Este salto ha dado lugar a lo que hoy se conoce como modelos de visión artificial embebida.

En esencia, se trata de arquitecturas de redes neuronales optimizadas para funcionar en dispositivos con muy poca memoria RAM (a menudo menos de 1 MB) y con procesadores de bajo consumo. Para lograrlo, se aplican técnicas como la cuantización (reducir la precisión de los pesos de 32 a 8 bits, por ejemplo), la poda de capas (eliminar conexiones o filtros que apenas aportan al rendimiento) o la compresión de parámetros [14]. Todo ello permite que modelos que originalmente ocupaban cientos de megabytes se reduzcan a unos pocos, sin perder demasiada precisión.

Entre los modelos más utilizados en este contexto destacan MobileNet [15], o YOLO-tiny [16], ya que son los que mejor se ajustan a los límites de memoria y velocidad. El caso de MobileNetV2 es bastante claro: ocupa apenas 4 MB, mientras que una red más pesada, como ResNet50, se dispara fácilmente por encima de los 100 MB. Esa diferencia es lo que hace posible que placas como la Portenta H7 puedan procesar una inferencia en decenas de milisegundos; una cifra que habría parecido ciencia ficción hace apenas diez años.

Lo curioso es que ya no hablamos de pruebas de laboratorio, sino de proyectos aplicados. En obras de construcción, por ejemplo, se ha utilizado YOLO-tiny para comprobar de manera automática si los trabajadores llevan chaleco [17]. En fábricas, algunos ensayos han servido para confirmar si un operario lleva puestos los guantes de seguridad. Y en agricultura empiezan a aparecer soluciones que cuentan frutos o detectan plagas usando redes reducidas, todo esto

directamente en sensores de campo, sin depender de una conexión estable a internet [18]. Al final, estos casos demuestran que la visión embebida ya no es solo un tema de investigación, se ha convertido en una herramienta práctica que se está usando de verdad.

Las ventajas de este enfoque son claras:

- Baja latencia: las decisiones se toman en el momento, sin esperar a un servidor externo.
- Menor dependencia de la red: el sistema funciona incluso sin conexión.
- Privacidad: las imágenes no salen del dispositivo, evitando exponer datos sensibles.
- Eficiencia energética: estos modelos consumen muy poca energía, lo que los hace aptos para dispositivos alimentados por batería.

Sin embargo, también existen limitaciones. La precisión de los modelos embebidos suele ser algo menor que la de sus equivalentes de gran tamaño. Además, requieren datasets bien diseñados y equilibrados, porque no tienen capacidad de compensar carencias en los datos con un exceso de parámetros. Tampoco existe todavía una estandarización clara, cada fabricante ofrece sus propias librerías, y trasladar un modelo de un hardware a otro puede implicar modificaciones importantes. Por último, no hay que olvidar los riesgos de seguridad, cualquier dispositivo embebido es susceptible de accesos físicos no autorizados o de ataques que intenten manipular los datos [19].

En el caso de este proyecto, los modelos de visión embebida han sido fundamentales. El dataset se capturó manualmente con la cámara de la Portenta, se entrenó en Edge Impulse y, tras varias iteraciones, se exportó un modelo optimizado y compatible con el microcontrolador. Se eligió una resolución relativamente baja (136×136 píxeles) como equilibrio entre precisión y velocidad, ya que resoluciones mayores no ofrecían mejoras proporcionales en este hardware. Con esta configuración, nuestro sistema es capaz de realizar la inferencia en tiempo real y decidir en el mismo instante si un operario lleva casco o no.

En definitiva, los modelos de visión artificial embebida representan la unión entre la inteligencia artificial y los sistemas IoT. Su evolución ha permitido que dispositivos del tamaño de una tarjeta de crédito sean capaces de tomar decisiones que antes requerían servidores completos. Este proyecto se inscribe en esa tendencia, demostrando que es posible implementar un sistema de seguridad laboral basado en visión embebida, con resultados prácticos y potencial de crecimiento hacia otros equipos de protección personal en el futuro.

2.4. Tecnologías y dispositivos hardware

2.4.1. Portenta H7

La Arduino Portenta H7 es una de las placas más potentes dentro del ecosistema Arduino, pensada para aplicaciones profesionales de IoT e inteligencia artificial en el borde. Su principal característica es que incorpora un microcontrolador STM32H747 con doble núcleo: un Cortex-M7 que puede alcanzar hasta 480 MHz y un Cortex-M4 a 240 MHz. Esta arquitectura permite ejecutar tareas en paralelo y aprovechar el núcleo más potente para el procesamiento intensivo, como la inferencia de modelos de visión artificial, mientras que el otro gestiona operaciones auxiliares o de bajo nivel [20].

Figura 3. Portenta H7

Fuente: https://www.arduino.cc/pro/hardware-product-portenta-h7/

La placa cuenta con 8 MB de SDRAM y 16 MB de memoria flash, lo que la diferencia de otros microcontroladores más limitados y la convierte en una opción adecuada para ejecutar modelos de TinyML. Dispone de conectividad integrada (Wi-Fi y Bluetooth Low Energy), que facilita su uso en aplicaciones industriales conectadas. Además, a través de sus conectores de alta densidad es posible añadir shields especializados, como la Vision Shield utilizada en este proyecto.

Otro aspecto relevante es la compatibilidad con distintos entornos de programación. Aunque forma parte del ecosistema Arduino, la Portenta H7 puede programarse en C++ con el IDE clásico, o bien en MicroPython y OpenMV, lo que la hace muy versátil para tareas de visión artificial. Esta flexibilidad ha sido clave en el proyecto, ya que permitió capturar imágenes, entrenar el modelo y después integrarlo con la lógica del lector RFID y la sincronización en la nube usando un lenguaje conocido.

La Portenta está pensada para funcionar en entornos industriales exigentes: soporta alimentación mediante USB-C y PoE, incluye protección frente a variaciones de tensión y puede trabajar en un rango amplio de temperaturas. Todo ello la convierte en un dispositivo robusto, apto tanto para prototipos de laboratorio como para proyectos que busquen acercarse a un despliegue real en la industria.

En este trabajo, la elección de la Portenta H7 estuvo motivada por ese equilibrio entre potencia de cómputo, conectividad y soporte de herramientas de TinyML. Su capacidad para ejecutar inferencias en tiempo real, junto con la posibilidad de añadir módulos como la cámara de la Vision Shield, ha hecho posible implementar un sistema de detección de EPIs completamente autónomo, sin depender de servidores externos para procesar las imágenes.

2.4.2. Vision Shield LoRa

La Arduino Portenta Vision Shield LoRa es una expansión diseñada específicamente para complementar a la Portenta H7 con funciones de visión artificial y conectividad adicional. Su elemento más destacado es una cámara integrada (OV5640, con sensor de 5 megapíxeles), que permite capturar imágenes directamente en la placa sin necesidad de hardware externo. Esta cámara admite distintas resoluciones y modos de captura, lo que facilita adaptar el flujo de datos a los límites de memoria y procesamiento de la Portenta [21].

Figura 4. Vision Shield LoRa

Fuente: https://www.arduino.cc/pro/hardware-product-portenta-vision-shield/

Además de la cámara, la Vision Shield incorpora un micrófono digital de doble canal, pensado para proyectos de reconocimiento de sonido o entornos multimodales donde se combinen imagen y audio. Aunque en este proyecto no se ha utilizado, constituye un recurso útil para ampliar la funcionalidad en aplicaciones futuras.

Otro aspecto importante de este shield es que ofrece conectividad LoRa gracias a un módulo SX1276, lo que teóricamente permitiría enviar datos a largas distancias con bajo consumo energético. Sin embargo, en este trabajo esa parte no se ha utilizado finalmente debido a la falta de cobertura adecuada de gateways en la zona de pruebas. En su lugar, se optó por la conexión Wi-Fi de la propia Portenta para sincronizar los datos en la nube. Aun así, la posibilidad de contar con LoRa convierte a esta placa en una opción interesante para entornos industriales donde la red Wi-Fi no esté disponible.

La Vision Shield se conecta a la Portenta H7 a través de los pines de alta densidad y se integra perfectamente tanto a nivel de hardware como de software. La compatibilidad con OpenMV facilita la captura y preprocesamiento de imágenes, lo que ha sido fundamental para construir el dataset y ejecutar el modelo de detección directamente en el dispositivo.

En este proyecto, la Vision Shield ha cumplido principalmente el papel de cámara embebida, permitiendo capturar las imágenes necesarias para entrenar el modelo y, más adelante, procesar en tiempo real las fotos que se toman al pasar una tarjeta RFID. El hecho de disponer de una cámara integrada y optimizada para la Portenta simplifica mucho el diseño, evitando tener que añadir periféricos externos que podrían complicar la integración.

2.4.3. Lector RFID

El sistema de identificación de operarios se apoya en un lector RFID [22] compacto de control de acceso, diseñado para entornos industriales y exteriores. Se trata de un dispositivo metálico, de tamaño reducido (25 × 22 × 20 mm) y con certificación de protección IP65, lo que le permite resistir polvo y salpicaduras de agua, algo importante en fábricas o talleres donde las condiciones no siempre son limpias o secas [22].

Figura 5. Lector de tarjetas RFID

Este lector soporta dos tecnologías habituales en la industria:

Tarjetas de baja frecuencia (125 KHz, tipo EM Marine/EM4100), que ofrecen una identificación sencilla y económica.

Tarjetas de alta frecuencia (13,56 MHz, MIFARE ISO14443A), más seguras y con capacidad de almacenar información adicional.

En nuestro proyecto usaremos tarjetas de baja frecuencia (125 KHz), [23] la lectura se realiza a una distancia de entre 2 y 8 cm, rango corto que obliga al trabajador a acercar la tarjeta de forma intencionada. Esto evita lecturas accidentales y refuerza el control de accesos.

Figura 6. Tarjetas RFID 125 KHz

Para la comunicación con la Portenta H7, el lector emplea el protocolo Wiegand 26/34, muy extendido en sistemas de control de acceso. Este protocolo transmite el identificador de la tarjeta

en forma de secuencia de pulsos digitales, lo que facilita la integración con microcontroladores y sistemas embebidos. En este proyecto, la Portenta interpreta dichos pulsos mediante un módulo software que decodifica el formato Wiegand y asocia cada tarjeta con un operario registrado en la base de datos.

El protocolo utilizado es Wiegand 26, es uno de los más comunes en sistemas de control de acceso (puertas con tarjeta, lectores RFID, etc.). Fue originalmente creado para dispositivos con tarjetas magnéticas, pero sigue muy usado con tarjetas RFID, tags, y lectores de huella o teclado.

Es un protocolo de comunicación unidireccional basado en hardware, que transmite datos en forma binaria a través de dos líneas de datos llamados Data0 y Data1. El número "26" se refiere al número total de bits que se transmiten por evento. En reposo, ambas líneas están en estado alto (‘1’ lógico).

Un pulso de nivel bajo en D0 significa un bit recibido con valor 0.

Un pulso de nivel bajo en D1 significa un bit recibido con valor 1.

En total se reciben 26 bits en este orden:

MSB - Bit 1 (1)	Bit 2..9 (8)	Bit 10..25 (16)	LSB - Bit 26 (1)
Paridad impar	siteCode	userCode	Paridad par

- P1 → bit de paridad impar calculado sobre los 12 primeros bits (del bit 2 al bit 13).
- Código de sitio (Facility Code) → identifica la instalación (0 a 255).
- Número de tarjeta (Card Number) → identifica la tarjeta (0 a 65535).
- P2 → bit de paridad par calculado sobre los 12 últimos bits (del bit 14 al bit 26).

Ejemplo de análisis del protocolo “Wiegand 26” con analizador lógico

Utilizando un analizador lógico marca Saleae con frecuencia de muestre de 4Mhz la lectura de la etiqueta ‘tag’ 148 19828 nos muestra el siguiente cronograma para las líneas D0 y D1, por defecto se encuentran en estado alto “1”

La trama recibida es, por lo tanto:

D0	--00-0-000-00--0-0---0-00-
D1	11--1-1---1--11-1-111-1--1
Trama	11001010001001101011101001

Trama	1 10010100 0100110101110100 1
	pi 148 19828 pp

El 1º bit de paridad corresponde a la paridad impar de los 12 primeros bits de datos

```
pi(100101000100)= 1
```

El último bit, corresponde a la paridad par de los 12 últimos bits de datos

```
pp(110101110100)= 1
```

En cuanto a la frecuencia de la transmisión, se produce con una ventana de trama de aproximadamente 2,1 ms, lo que supone un tiempo total de transmisión de los 26 bits de 54,6 ms, por lo que escogeremos para nuestra rutina de lectura un TimeOut de 50 ms, de esta forma si después de recibir el primer bit de la trama, pasan más de 50 ms el buffer de lectura se limpiará automáticamente.

En conjunto, el lector RFID actúa como la primera capa de validación: permite identificar quién intenta acceder a la sala de máquinas antes de comprobar, a través de visión artificial, si lleva los equipos de protección individual requeridos.

Capítulo 3: Diseño del sistema

3.1. Diagrama general del sistema

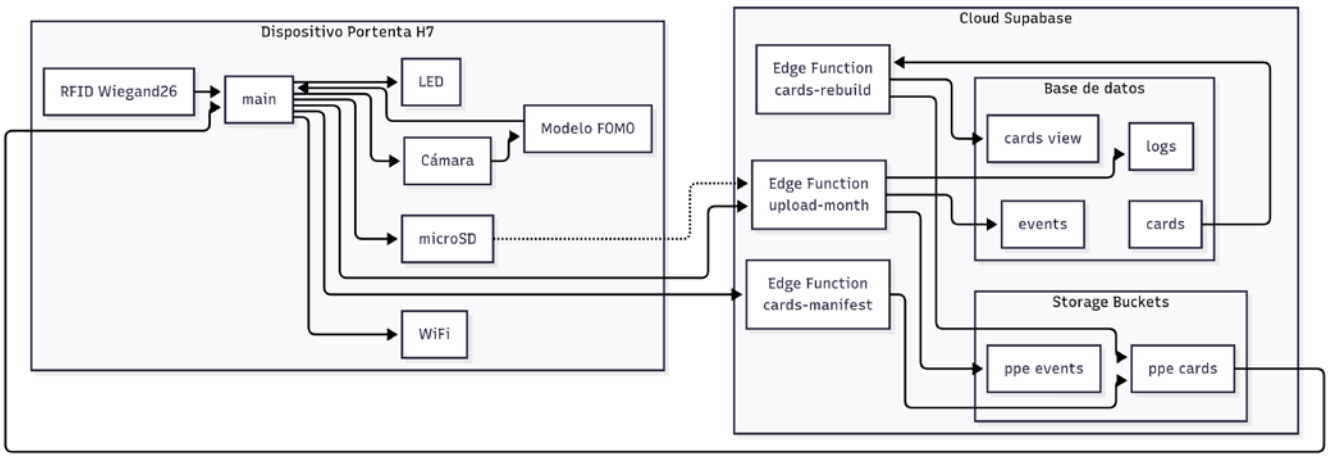

Figura 7. Diagrama general del sistema

El sistema inicial no se mantuvo tal y como se había planteado, sino que fue cambiando en cada versión. A medida que aparecían problemas en una etapa, se aplicaban modificaciones para poder avanzar. En unos casos las correcciones respondían a limitaciones técnicas, mientras en otros, a requisitos que se incorporaron más tarde. Este modo de trabajo hizo posible comprobar de manera progresiva el comportamiento del prototipo en situaciones reales. Así se fue ganando experiencia y corrigiendo errores, sin tener que asumir toda la complejidad del proyecto desde el principio.

Primera versión: la Portenta conectada al portátil.

El punto de partida fue lo más básico posible: la placa Portenta H7 conectada directamente al ordenador mediante un cable USB-C. Esa conexión servía a la vez para alimentar la placa y para acceder desde el IDE de OpenMV. Con este montaje inicial se desarrolló un pequeño programa que capturaba automáticamente una fotografía cada tres segundos. Así se fueron obteniendo las primeras imágenes que acabarían formando parte del dataset para entrenar el primer modelo de detección de casco, que posteriormente se descartó al usar la segunda versión del sistema de la que hablare a continuación.

Esta configuración cumplió su función como prototipo, pero era poco práctica. El hecho de no controlar la captura de imagen limitaba mucho su uso y no siempre se ajustaba a lo que necesitaba: a veces quería varias fotos seguidas, y otras prefería esperar a tener una mejor postura o iluminación. En definitiva, era útil para arrancar, pero no suficiente para un trabajo más controlado.

Segunda versión: añadir un pulsador para la captura de imágenes.

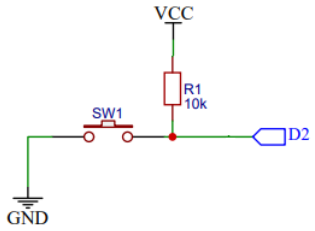

Figura 8. Conexión de un pulsador al puerto GPIO D2 de la Portenta

La siguiente evolución fue introducir un botón físico conectado a la Portenta. Con él, la captura de imágenes dejó de ser automática y pasó a estar bajo control directo. Bastaba con pulsar el botón conectado a uno de los GPIO de la portenta para que la cámara tomase una foto y la guardara en la tarjeta microSD.

Este cambio, aunque aparentemente pequeño, supuso una gran mejora. Por primera vez el sistema dejaba de ser un mero dispositivo esclavo del PC y empezaba a incluir interacción física propia. La Portenta ya podía funcionar de manera más autónoma, y el proceso de captura del dataset se volvió mucho más eficiente: ya no hacía falta descartar tantas imágenes innecesarias y era posible elegir el momento exacto de la captura.

Tercera versión: integración del lector RFID

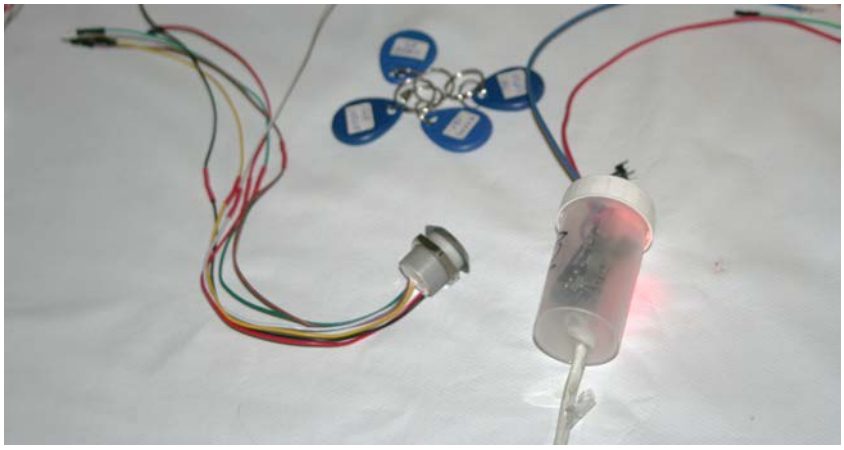

Figura 9. Lector RFID, tarjetas y FA 12V.

Una vez dominada la parte de captura de imágenes, llegó el momento de añadir identificación de usuarios. Para ello se integró un lector RFID de control de acceso. Este lector, de tamaño reducido y con carcasa metálica, funciona tanto con tarjetas de baja frecuencia (125 kHz, tipo EM Marine) como con tarjetas MIFARE (13,56 MHz). Se alimenta de forma independiente con 12 V

y transmite los datos a la Portenta mediante el protocolo Wiegand-26, muy utilizado en sistemas de control de acceso.

Con esta incorporación, el sistema cambió de nivel. Ya no se trataba únicamente de tomar fotos: ahora podía reconocer qué trabajador había pasado su tarjeta en el punto de control. La Portenta recibía el código binario de la tarjeta, lo procesaba y lo comparaba con una lista de autorizaciones almacenada localmente en un CSV. Así, cada evento quedaba asociado a un usuario concreto, lo que abría la puerta a registrar accesos reales y no simples pruebas.

Cuarta versión: sistema autónomo e independiente.

En esta etapa el objetivo fue conseguir que el sistema pudiera operar de forma continua sin depender en absoluto de un ordenador externo. La Portenta H7 ya ejecutaba toda la lógica desde las versiones anteriores -lectura de tarjeta, captura de imagen e inferencia local del modelo de detección- pero todavía estaba limitada por la alimentación a través del puerto USB del portátil y por la ausencia de una interfaz visual clara.

Para superar esas limitaciones, se incorporó una fuente de alimentación externa, lo que permitió instalar la Portenta como un dispositivo fijo, sin necesidad de estar conectada a un PC. También se añadieron LEDs de estado como mecanismo de señalización: azul al detectar una tarjeta, verde o rojo según el resultado de la verificación de casco, y parpadeos en caso de error. Gracias a estos indicadores, el sistema podía usarse de manera directa en un entorno real, sin depender de pantallas o del IDE para interpretar lo que estaba ocurriendo.

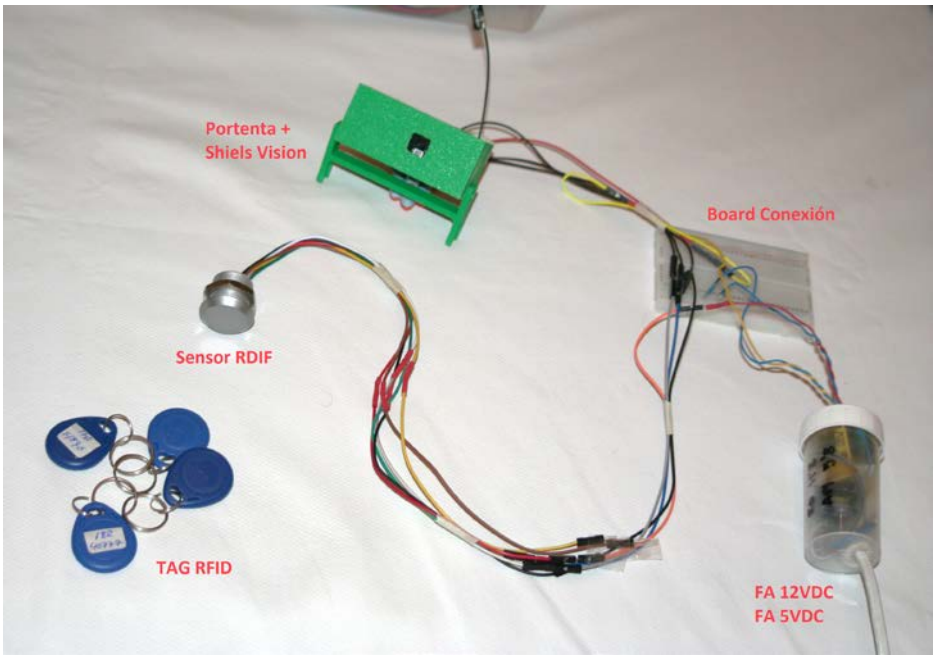

Figura 10. Conexión Sistema independiente con FA, RFID, Portenta H7.

En paralelo, los eventos comenzaron a registrarse en archivos CSV dentro de la tarjeta microSD, lo que facilitaba llevar un control histórico sin intervención del usuario. Con estos cambios, el sistema pasó de ser un prototipo de laboratorio conectado a un ordenador, a un dispositivo compacto y totalmente autónomo, listo para probarse en condiciones similares a las de un entorno industrial.

Quinta versión (última): integración con la nube.

El último paso en la evolución del sistema fue añadir un componente en la nube que complementase el funcionamiento local de la Portenta. Hasta este punto, el dispositivo ya podía trabajar de manera completamente autónoma, pero seguía siendo un sistema cerrado: registraba eventos en la microSD y validaba tarjetas desde un archivo local. Con la incorporación de la capa cloud, el proyecto dio un salto hacia un sistema distribuido y mucho más flexible.

En esta versión, la Portenta se conecta por Wi-Fi a un servicio en la nube basado en Supabase [24], que cumple dos funciones principales. La primera es la sincronización de eventos: cada vez que un operario pasa su tarjeta y se ejecuta la verificación de EPIs, la Portenta genera un registro que, además de guardarse en la microSD, se sube a la base de datos remota. De esta forma, los accesos quedan centralizados y disponibles para su consulta desde cualquier lugar, facilitando la trazabilidad y el análisis posterior.

La segunda función es la gestión remota de tarjetas. En lugar de tener que actualizar manualmente la lista de usuarios autorizados en el dispositivo, el sistema descarga periódicamente desde la nube un archivo cards.csv generado automáticamente a partir de la base de datos. Esto permite que un administrador pueda dar de alta, modificar o revocar accesos directamente desde Supabase, sin necesidad de intervenir físicamente en el dispositivo.

Con esta quinta versión, el sistema ya no depende únicamente de su hardware local: se convierte en un sistema híbrido, donde la Portenta se encarga del procesamiento en el borde (lectura de tarjeta, detección visual, control de acceso) y la nube amplía las capacidades de almacenamiento y administración. De esta manera, se logra una solución más completa, escalable y cercana a un escenario industrial real, donde los datos y la gestión centralizada son imprescindibles.

Reflexión sobre la evolución.

La opción de partir de una versión simple para ir evolucionando fue lo que verdaderamente me permitió avanzar. La primera fue simplemente arrancar la cámara de la Portenta y empezar a guardar imágenes, nada más, pero con eso ya podía trabajar. Con la segunda, añadí el pulsador y fue un cambio grande: ya podía decidir cuándo hacer la foto y el dataset salió mucho más controlado.

La tercera fue distinta, porque ahí ya entró el lector RFID. Eso hizo que el sistema empezara a parecerse a lo que se necesita en un control de acceso real, con usuarios identificados. Después vino la cuarta, que fue conseguir que todo funcionara sin depender de la conexión con Pc, con los LEDs mostrando lo que pasaba. Ahí ya se sentía como algo que podía estar en marcha sin que yo tuviera que estar al lado con el portátil.

Y la última, la quinta, fue añadir la nube. Para que los registros no se quedaran solo en la tarjeta de memoria y que las tarjetas de acceso se pudieran gestionar también desde fuera. Con eso cerraba el círculo.

Cada paso tenía un problema claro que resolver: primero no quería depender del PC, luego necesitaba controlar la captura, después identificar a cada usuario, más tarde que funcionara solo, y al final que todo pudiera consultarse y actualizarse desde la nube. Al final no fue un plan perfecto desde el inicio, fue más bien en método propuesta de mejora -> modifica -> probar, modificando los errores y amentando las prestaciones. Y eso hizo que el sistema terminara con más sentido del que parecía al principio.

3.2. Flujo de funcionamiento del sistema

El flujo del sistema comienza en el

momento en que un operario presenta su tarjeta en el lector RFID y termina cuando el evento queda registrado tanto en la microSD como en la nube. Los pasos se suceden de la siguiente manera:

1. Lectura de tarjeta:

El lector RFID envía el identificador en formato Wiegand-26, la Portenta H7 interpreta los 26 bits y extrae el site_code y el user_code correspondientes.

2. Comprobación de autorización:

El dispositivo consulta la lista local de accesos (cards.csv) que se encuentra en la microSD.

Si la tarjeta no aparece como autorizada, se activa el LED azul y se guarda un evento de acceso denegado en el CSV local.

Si la tarjeta está en la lista, el sistema continúa con la parte de visión.

3. Captura de imagen:

La cámara de la Vision Shield toma una foto del operario en el mismo instante en que se ha presentado la tarjeta.

4. Inferencia del modelo:

La Portenta ejecuta el modelo entrenado en Edge Impulse (formato FOMO) para decidir si la persona lleva casco. Se obtiene un resultado binario (casco / no casco) junto con un valor de confianza.

5. Señalización y registro local:

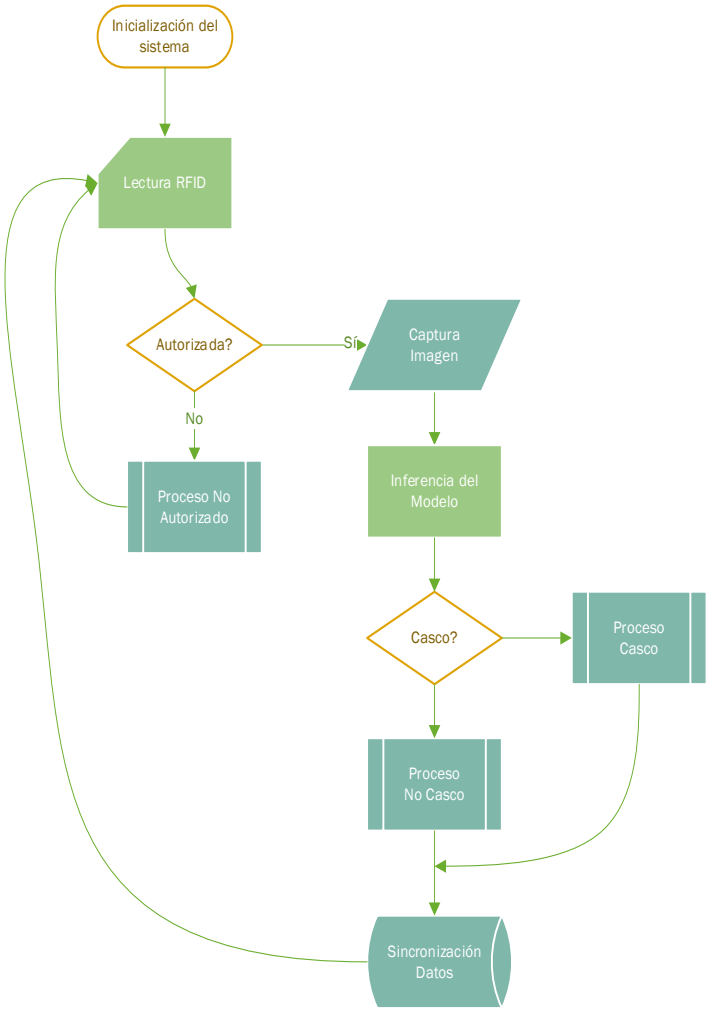

Figura 11. Flujo de funcionamiento del sistema

El sistema muestra el resultado con los LEDs: verde si se detecta casco, rojo si no. En paralelo, se añade una línea al archivo de eventos (events_YYYYMM.csv) en la microSD, con todos los datos: tarjeta, usuario, estado de autorización, resultado de la inferencia, puntuación y, si está configurado, la ruta de la imagen guardada como prueba.

6. Sincronización con la nube:

Después de procesar la tarjeta, la Portenta intenta subir los archivos del mes en curso (CSV y manifest) a Supabase mediante una Edge Function. No es un envío programado en intervalos fijos: se dispara tras cada evento, con un margen mínimo de varios segundos para evitar repeticiones seguidas. Así, los datos locales y la base de datos remota permanecen alineados de manera casi inmediata.

7. Actualización de tarjetas de acceso:

El archivo cards.csv también puede renovarse desde la nube. La Portenta comprueba la existencia de nuevas versiones al arrancar, cada vez que recupera conexión Wi-Fi y de forma periódica cada diez minutos. Si hay una versión más reciente, la descarga y sustituye el archivo local, de modo que la lista de accesos se mantiene siempre al día.

3.3. Requisitos funcionales y no funcionales

A partir de los objetivos definidos en el apartado 1.2, se formularon los requisitos del sistema. Estos describen, de forma concreta y verificable, las condiciones mínimas que debe cumplir el prototipo para considerar que aquellos objetivos se han alcanzado. Más abajo se presentan los requisitos funcionales y no funcionales, junto con los criterios que permitirán su validación experimental.

3.3.1 Requisitos funcionales

- RF1. Identificación. El sistema debe identificar al operario mediante la lectura de una tarjeta RFID (Wiegand-26) y obtener site_code y user_code.
- RF2. Captura de imagen. El sistema debe capturar automáticamente una imagen en el momento del intento de acceso válido.
- RF3. Inferencia embebida. El sistema debe ejecutar un modelo embebido que clasifique entre las clases “casco” y “no casco”, con tiempo de inferencia ≤ 500 ms.
- RF4. Feedback. El sistema debe mostrar el resultado de la verificación al usuario mediante señales luminosas: LED verde (casco correcto), rojo (sin casco) o azul (no autorizado).
- RF5. Registro local. Cada intento de acceso debe guardarse en un archivo CSV con: identificador de tarjeta, fecha, hora, resultado de la detección.
    - RF6. Sincronización. Los registros deben sincronizarse con la base de datos en la nube cuando exista conectividad, confirmando la subida completa de cada fichero.
    -

3.3.2 Requisitos no funcionales

- RNF1. Latencia. El tiempo de respuesta entre el pase de la tarjeta y la señal luminosa debe ser ≤ 2000 ms
- RNF2. Operación sin red. El sistema debe ser capaz de registrar todos los eventos en ausencia de Wi-Fi y sincronizarse automáticamente al recuperarse la conexión, sin pérdida de datos.
- RNF3. Precisión del modelo. El modelo embebido debe alcanzar al menos F1 ≥ 0,85 (con precisión y recall ≥ 0,80) en condiciones de iluminación representativas.
- RNF4. Integridad de registros. Los archivos CSV deben almacenarse de forma trazable y verificable, usando hash SHA-256.
- RNF5. Estabilidad. El sistema debe mantener sesiones continuas de al menos 6 horas de funcionamiento sin bloqueos, fugas de memoria ni pérdidas de datos.

Objetivo	Descripción	Requisito asociado
O1	Detectar localmente en la Portenta H7 si un operario lleva casco o no, de forma instantánea.	RF2, RF3, RNF1, RNF3
O2	Integrar el lector RFID para identificar al operario antes de la comprobación visual.	RF1
O3	Sincronizar la captura de imagen con el evento RFID para evitar desfases.	RF2, RNF1
O4	Registrar localmente cada intento de acceso en la microSD en formato CSV (fecha, tarjeta, estado).	RF5, RNF4
O5	Sincronizar los registros con la nube (Supabase), guardando doble copia en bucket y tabla relacional.	RF6, RNF2, RNF4
O6	Garantizar funcionamiento offline, conservar eventos sin conexión y subirlos más tarde.	RNF2, RF5

Figura 12. Tabla de relación objetivo-requisito.

La tabla muestra cómo cada objetivo inicial (O1–O6) se apoya en uno o varios requisitos concretos. De esta manera, lo que en un principio eran metas generales queda traducido a condiciones medibles con requisitos funcionales y no funcionales. Gracias a esta relación, es posible comprobar en el capítulo de resultados si cada objetivo se alcanzó realmente.

3.4. Limitaciones hardware

El montaje se hizo sobre una Portenta H7 con su Vision Shield LoRa. Es un hardware interesante dentro del mundo embebido: pequeño, versátil y con bastante potencia para su tamaño. Aun así, deja ver limitaciones que condicionaron el diseño y que conviene señalar.

La memoria es el primer freno. Solo admite modelos ligeros y cuantizados; redes profundas quedan fuera de juego. Esto obligó a trabajar con arquitecturas optimizadas y a reducir preprocesado. La cámara integrada responde bien cuando hay luz suficiente, pero en interiores poco iluminados aparecen fotos movidas y ruido. Para salvar esta limitación se recurrió a capturas disparadas en el momento justo en lugar de hacerlo con temporizador.

Respecto al almacenamiento, la microSD funciona, pero hay que tratarla con cuidado. Si se corta la energía en plena escritura, el archivo puede corromperse. Por eso se fijó un tope de tamaño en los CSV, se rotaron de manera mensual y se aplicó verificación con hash. Algo parecido pasa con la alimentación por USB: es cómoda, sí, pero sensible a cables largos o a puertos con poca entrega de corriente.

El feedback al usuario quedó reducido a tres LEDs de colores. Cumplen para un prototipo, porque indican de forma rápida si el acceso está permitido o no, pero en un entorno industrial real se necesitarían mecanismos más ricos: pantallas, señales acústicas o integración con torres de luz. Con la conexión Wi-Fi ocurre algo similar. Funciona, aunque con caídas y variaciones de latencia; de ahí que la lógica se pensara para aguantar sin red y sincronizar más tarde.

El lector Wiegand también trajo lo suyo. Sin filtrado, la misma tarjeta podía registrar varios accesos si se quedaba pegada al lector. Para evitarlo se definió una ventana de antirrebote de seis segundos. Y queda otro punto: el reloj. La placa no conserva la hora al apagarse, lo que obliga a sincronizarse con NTP en cada arranque; si no hay conexión, se mantiene la última referencia válida.

Por último, la robustez física. Ni la Portenta ni el Vision Shield están pensados para polvo, vibraciones o humedad. Sin una carcasa adecuada ni conectores de tipo industrial, el montaje difícilmente soportaría un entorno de producción.

En conjunto, todas estas limitaciones no impiden demostrar la viabilidad del sistema, pero sí marcan su alcance: es un prototipo válido como prueba de concepto, aunque todavía lejos de un despliegue directo en planta sin reforzar el hardware.

Capítulo 4: Implementación

4.1. Integración de componentes

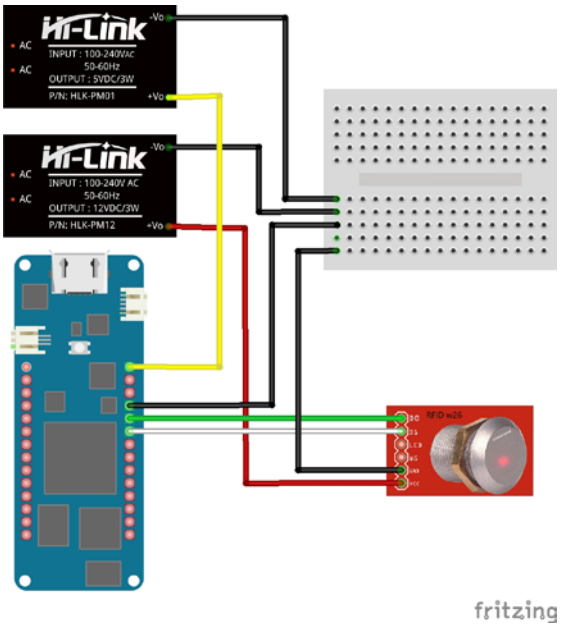

Figura 13. Esquema de conexión

La primera fase de la implementación consistió en la conexión física de los distintos módulos al núcleo del sistema, la Portenta H7. En la Figura 14 se muestra el esquema eléctrico utilizado, donde se aprecian tanto las líneas de alimentación como las de datos.

El sistema parte de dos módulos de alimentación que permiten transformar la corriente alterna en tensiones continuas seguras y estables para los periféricos. El módulo conectado a la Portenta H7 proporciona 5’3 V, ya que si se quedaba en 5V (medidos) no era suficiente para el correcto funcionamiento de la placa, mientras que el módulo conectado al RFID suministra 12 V. Ambos comparten la referencia de masa (GND), unificando el retorno de corriente y garantizando una comunicación coherente entre los dispositivos.

En cuanto a la conexión de datos, el lector RFID utilizado trabaja bajo el protocolo Wiegand-26, que transmite la información mediante dos líneas digitales diferenciadas: D0 y D1. Estas señales llegan directamente a dos pines GPIO de la Portenta, configurados como entradas con interrupción para poder capturar los pulsos de forma precisa. Además de estas líneas de datos, el módulo RFID requiere alimentación (VCC a 12 V y GND compartido) y cuenta con un pin para el control de LED, que en esta implementación se mantiene vinculado al estado del propio lector, sirviendo como indicador visual de lectura.

La Portenta, a su vez, recibe alimentación desde el módulo de 5 V, pero internamente regula esta tensión a 3,3 V para el funcionamiento de su microcontrolador y periféricos integrados. En el esquema puede observarse cómo tanto el bus de 5 V como el de 12 V se distribuyen a través de una placa de prototipado, simplificando la conexión de los distintos módulos y permitiendo futuras ampliaciones.

En conjunto, este esquema asegura una separación clara de las líneas de potencia y de señal, al mismo tiempo que mantiene un punto común de referencia en GND. Con ello, la Portenta queda correctamente alimentada y es capaz de recibir los datos del lector RFID sin riesgo de niveles de tensión incompatibles. Este montaje constituye la base sobre la que se añadió posteriormente la cámara integrada en el Vision Shield y los periféricos de salida (LEDs de estado), completando la fase de integración hardware.

4.2. Captura del dataset

El conjunto de imágenes sobre el que se entrenó el modelo no se obtuvo de manera automática, sino a partir de varias pruebas iniciales. Algo de lo que me di cuenta muy pronto fue que el entorno tenía un peso enorme en la calidad de las capturas: la iluminación cambiaba según el lugar, el fondo no era uniforme y eso hacía que las imágenes resultaran muy irregulares. A esta falta de control se sumaba el uso de un temporizador que tomaba una fotografía cada pocos segundos. El sistema disparaba incluso cuando había ligeros movimientos, y al alargar la cámara la exposición en interiores, muchas fotos terminaban borrosas. El resultado fue un primer dataset heterogéneo y con bastantes imágenes descartadas.

Figura 14. Ejemplo del primer dataset (defectuoso)

Para evitarlo, se pasó a un sistema de disparo manual mediante un pulsador físico conectado a la placa. Con esta solución, cada imagen se tomaba en el momento exacto en que el operario estaba estático y correctamente encuadrado. De este modo se redujo de forma notable el número de fotos inservibles y se consiguió un dataset más limpio y representativo.

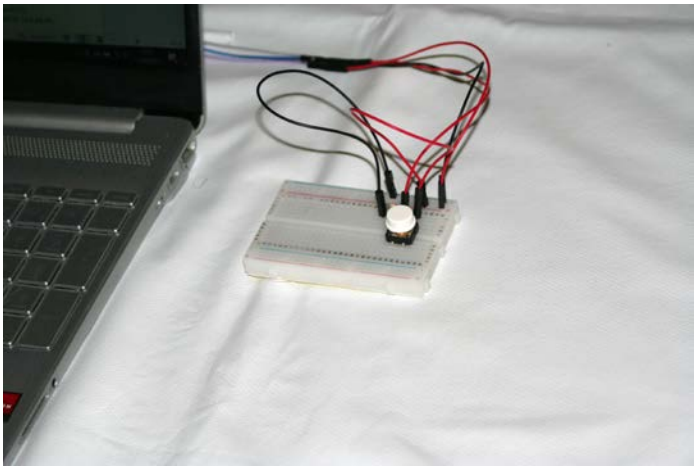

Figura 15. Foto de cómo se implementó el pulsador

Otro aspecto fue la consistencia del encuadre. Para lograrla se diseñó una pequeña caja de soporte en 3D, que permitía mantener fija la Portenta sobre la pantalla de un portátil. Así se controlaba la altura, la inclinación y el campo de visión, lo que aseguraba que todas las tomas tuvieran un ángulo prácticamente idéntico. Este detalle facilitó después el entrenamiento, ya que el modelo no tenía que lidiar con variaciones innecesarias.

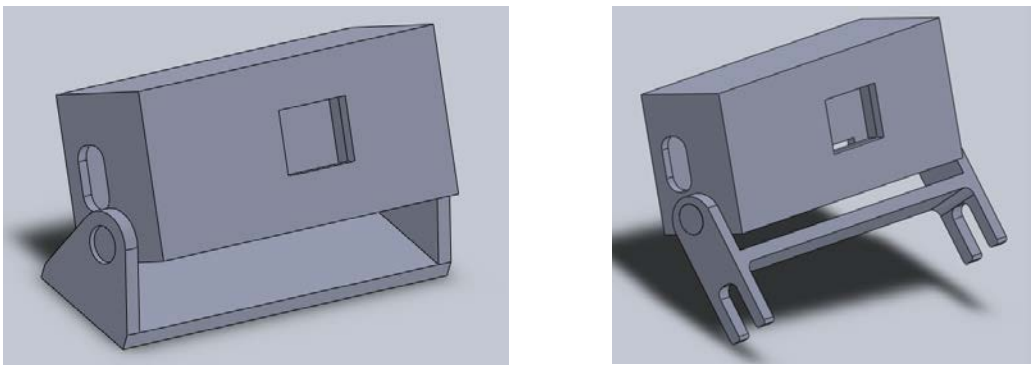

Figura 16. Imágenes del diseño de la caja y los soportes para su impresa en 3D.

Figura 17. Imágenes del sistema de captura con la caja impresa en 3D.

También se cuidó la iluminación y el fondo: se eligió un escenario neutro y con luz uniforme para que la red se centrase en las diferencias reales —presencia o ausencia de casco— y no en sombras o colores de fondo. Cuando se detectaban fallos de clasificación durante el entrenamiento, se añadían nuevas imágenes que reforzaban esos casos, siguiendo un proceso iterativo de captura-entrenamiento-revisión.

Figura 18. Ejemplo del segundo dataset (ejemplo con casco).

En conjunto, las decisiones sobre el método de disparo, el soporte físico y el control del entorno tuvieron un impacto decisivo: redujeron la proporción de imágenes descartadas, aumentaron la homogeneidad del dataset y facilitaron que el modelo convergiera con menos iteraciones.

Figura 19. Ejemplo del segundo dataset (ejemplo sin casco).

4.3. Entrenamiento del modelo en EdgeImpulse

El entrenamiento del modelo de detección se realizó en la plataforma Edge Impulse, que permitía integrar de forma sencilla las imágenes capturadas con la Portenta y generar un modelo optimizado para ejecutarse en la propia placa.

4.3.1 Configuración inicial

El primer entrenamiento se llevó a cabo con un dataset reducido de aproximadamente 500 imágenes, repartidas de forma equilibrada entre las dos clases (casco y nocasco), con un 50 % en cada categoría. Edge Impulse gestionó automáticamente la división en 80 % para entrenamiento y 20 % para validación, lo que permitía evaluar de manera rápida el comportamiento del modelo en un conjunto independiente de imágenes.

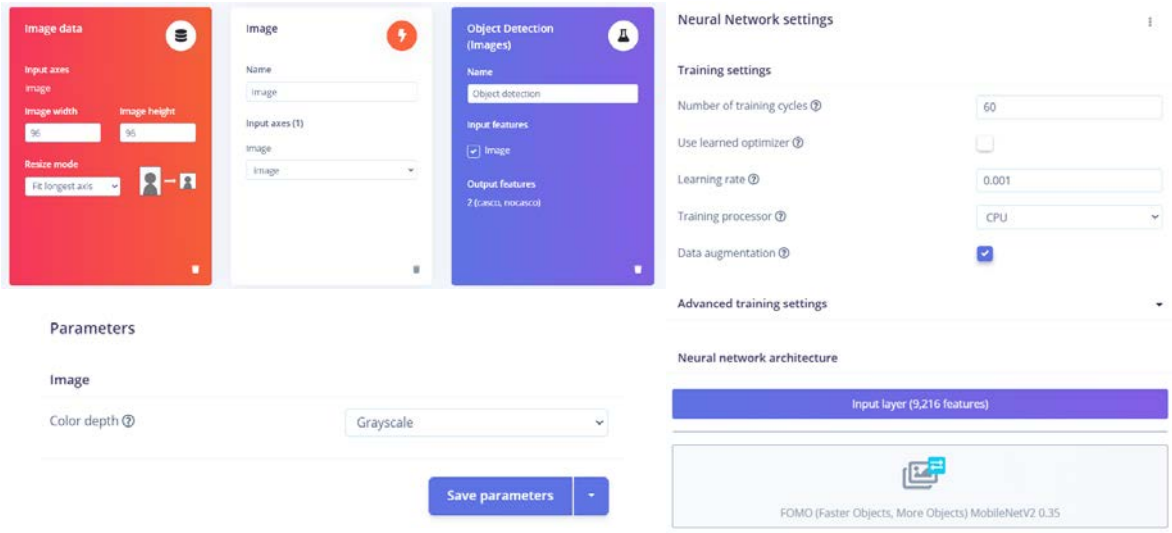

Figura 20. Configuración inicial en Edge Impulse

El impulse se configuró con resolución de 96×96 píxeles, en escala de grises, y el bloque de detección se basó en la arquitectura FOMO con MobileNetV2. Se entrenó con 60 épocas, una tasa de aprendizaje de 0,001 y un batch size de 32, sin aplicar aún técnicas de cuantización o ajustes avanzados. Con esta configuración inicial se obtuvo un primer modelo funcional, aunque con limitaciones de precisión y cierta sensibilidad a condiciones de iluminación variables.

4.3.2 Configuración final

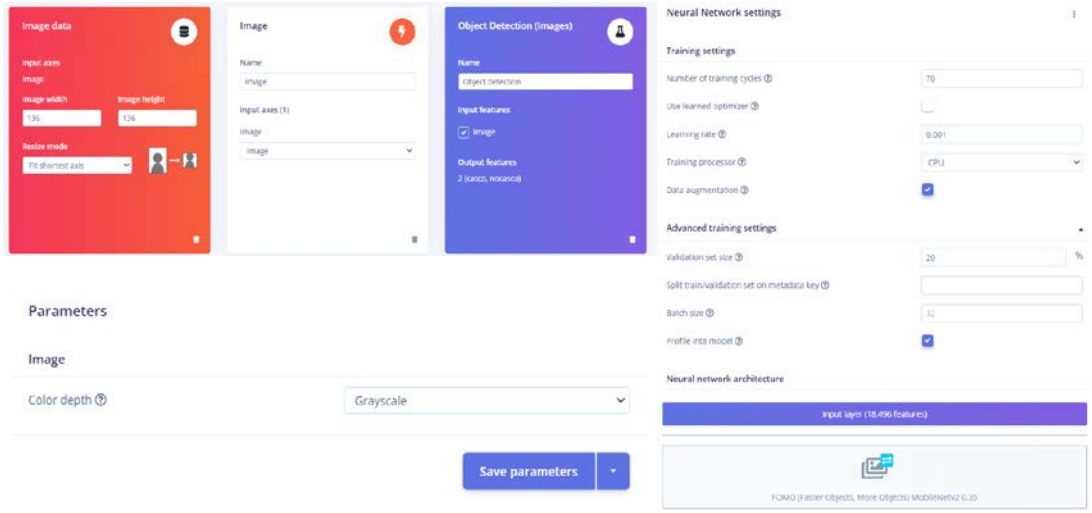

Figura 21. Configuración final en Edge Impulse

Tras sucesivas iteraciones y la incorporación de nuevas tandas de imágenes, el dataset final alcanzó unas 1650 imágenes, manteniendo el mismo balance entre clases y la misma proporción para entrenamiento y validación. Este incremento en la cantidad de datos, unido a una mejor calidad de las capturas (mayor control del entorno, uso del pulsador y soporte físico para la cámara), permitió mejorar la robustez del modelo.

En la configuración final del impulse se aumentó la resolución a 136×136 píxeles, conservando la escala de grises para reducir el volumen de características y acelerar la inferencia en la Portenta H7. El bloque de detección continuó siendo FOMO MobileNetV2 0.35, pero el entrenamiento se amplió a 70 épocas, manteniendo la tasa de aprendizaje en 0,001 y el batch size en 32. Se habilitó el uso de data augmentation para introducir pequeñas variaciones en las imágenes y así mejorar la generalización del modelo, y se activó el perfilado int8, que reducía el tamaño del modelo y optimizaba la velocidad de ejecución en el dispositivo sin degradar significativamente la precisión.

Con esta configuración final se alcanzaron los mejores resultados en validación, demostrando que la combinación de un dataset más amplio y un ajuste cuidadoso de parámetros era clave para obtener un rendimiento satisfactorio en condiciones reales.

4.4. Programación de la lógica del sistema en la Portenta H7

La programación del sistema se dividió en módulos independientes, cada uno encargado de una parte concreta del flujo de funcionamiento. Esta modularidad permitió mantener el código más legible, facilitar pruebas por separado y reutilizar funciones en diferentes fases del desarrollo.

4.4.1 main.py

Es el archivo que arranca al encender la Portenta y, en cierto modo, manda sobre el resto. Nada más iniciar, configura la cámara y carga el modelo de Edge Impulse (el trained.tflite junto con sus etiquetas). También deja preparado el lector de tarjetas RFID mediante las interrupciones de los pines D13 y D14.

Cuando alguien acerca su tarjeta, el sistema recibe los 26 bits del protocolo Wiegand. Que se recibe mediante la configuración de dos vectores de interrupciçón asociados a los pind D13(Data0) y D14(Data1). Primero se comprueba que la trama es válida, y a partir de ahí se extraen los códigos de sitio y usuario. Con esos datos se consulta la ACL local para ver si está permitido el acceso. Si no lo está, se anota un evento de denegación, se enciende un LED azul y no se sigue adelante. Si sí está autorizado, el sistema pasa al segundo filtro: la verificación de casco. Para ello la cámara toma varias imágenes rápidas, el modelo procesa y devuelve si se ha detectado casco o no. El resultado se refleja con un LED verde (permitido) o rojo (denegado) y queda registrado junto con la evidencia correspondiente, la cual se sube a supabase en cada lectura del RFID.

En paralelo, el bucle principal también va llamando cada 10 minutos a la sincronización con Supabase para la actualización de la lista de tarjetas. Así, aunque lo fundamental ocurre en tiempo real cuando se lee una tarjeta, siempre hay procesos secundarios que mantienen el sistema al día.

4.4.2 storage_local.py

Todo lo que ocurre queda guardado en la microSD. No es simplemente un log, sino una pequeña base de datos en forma de CSV: cada mes se genera un fichero distinto (events_YYYYMM.csv) con la fecha, la hora, el código de tarjeta, si estaba autorizada, el resultado de casco, el valor de confianza del modelo y, si corresponde, la ruta a la foto guardada.

Además de los CSV, se genera un “manifest” JSON que resume cuántos eventos hay, el hash SHA-256 del fichero, la versión del firmware y la última fecha de actualización. Esto permite que, antes de enviar nada a la nube, se pueda comprobar que los datos están íntegros.

También aquí se gestiona el archivo cards.csv, donde están todas las tarjetas autorizadas. Este archivo es el que se carga en memoria cada vez que arranca el sistema o cuando se detecta una actualización desde la nube.

4.4.3 cloud_sync.py

Una vez los datos están en la microSD, toca enviarlos fuera. Para ello se usa un cliente muy ligero que monta una petición multipart/form-data. Dentro se adjuntan dos ficheros: el CSV de eventos del mes y su manifest correspondiente. Todo se manda a una función Edge desplegada en Supabase.

Para la autenticación se incluye una clave x-edge-key en la cabecera. El código está preparado para manejar incluso respuestas chunked (algo habitual en servidores HTTP), de forma que primero intenta parsear el JSON directo y, si no, reconstruye el cuerpo antes de procesarlo. El resultado siempre se devuelve como un objeto de estado, lo que permite al sistema decidir si reintentar o confirmar que la subida fue correcta.

4.4.4 cards_sync.py

El fichero con las tarjetas no es estático: se va actualizando según lo que haya en la base de datos central. El procedimiento es sencillo: en el inicio y cada diez minutos la Portenta consulta un manifest en Supabase (cards-manifest). Ese manifest incluye versión, hash y la URL del CSV real.

Si el hash o la versión no coinciden con lo que ya hay en local, se descarga el nuevo fichero en temporal, se comprueba la integridad y, si todo va bien, se reemplaza el antiguo. Una vez hecho, se recarga la ACL en memoria, de modo que la próxima tarjeta que se pase por el lector ya se valida contra la lista más reciente.

4.4.5 wifi_setup.py

Para conectarse a la Wi-Fi se usa un archivo de configuración guardado en /config/wifi.json. Este fichero puede contener la contraseña en texto claro o cifrada con AES-CBC, aprovechando que la clave se deriva del identificador único de la propia placa. De esta forma, aunque se extraiga la tarjeta SD, no se puede reutilizar la configuración en otro dispositivo.

Después de conectarse, el sistema sincroniza la hora con servidores NTP y la ajusta a la zona horaria de Madrid, incluyendo los cambios automáticos entre invierno y verano. Esto es importante porque las marcas de tiempo de los eventos deben ser consistentes y no depender de que el dispositivo se reinicie.

4.4.6 Compatibilidad con SSL y peticiones HTTP

El firmware de OpenMV utilizado en la Portenta H7 no incluye un soporte completo para conexiones TLS, lo que impedía realizar peticiones HTTPS directamente a servicios externos como Supabase. Para resolver esta limitación se diseñaron dos adaptaciones clave en el cliente embebido:

Implementación de ussl.py mínimo

Se añadió un módulo local que reexporta las funciones del paquete ssl como ussl, garantizando que import ussl no produzca errores. Este módulo incluye la función wrap_socket() con soporte opcional de Server Name Indication (SNI), necesaria para que los servidores de Supabase devuelvan el certificado correcto en el handshake TLS. De esta forma, se habilitó el cifrado punto a punto sin necesidad de modificar el firmware base.

Adaptación de urequests.py

- El cliente HTTP original de MicroPython fue extendido para:
- Incorporar el uso de wrap_socket con server_hostname, activando SNI.
- Forzar la cabecera Connection: close, evitando que sockets quedasen colgados en la Portenta.
- Calcular y añadir automáticamente Content-Length cuando se envían datos o JSON.
- Rechazar respuestas con Transfer-Encoding: chunked, que no están soportadas en OpenMV, y apoyarse en que las funciones Edge de Supabase devuelvan siempre respuestas con Content-Length.

Gracias a estas modificaciones, el dispositivo es capaz de realizar peticiones HTTPS válidas y estables a los endpoints de Supabase (Edge Functions y Storage), manteniendo compatibilidad con los certificados del servidor y evitando bloqueos en el cliente. Esta solución permitió cumplir con los requisitos de seguridad y robustez sin necesidad de recompilar ni alterar el firmware oficial de OpenMV.

Capítulo 5: Evaluación experimental y resultados

5.1. Resultados de validación y test

Antes de pasar a las pruebas prácticas sobre la Portenta H7, conviene detenerse en los resultados obtenidos “en papel”, es decir, aquellos que proporciona la propia plataforma de entrenamiento tras la última iteración del modelo. Estos valores son la primera referencia para evaluar si la red neuronal entrenada es lo bastante sólida como para integrarse en un dispositivo embebido. Aunque posteriormente se comprobó que la realidad siempre introduce matices, estas métricas iniciales ofrecen una visión clara del potencial del sistema.

Resultados del modelo en fase de validación

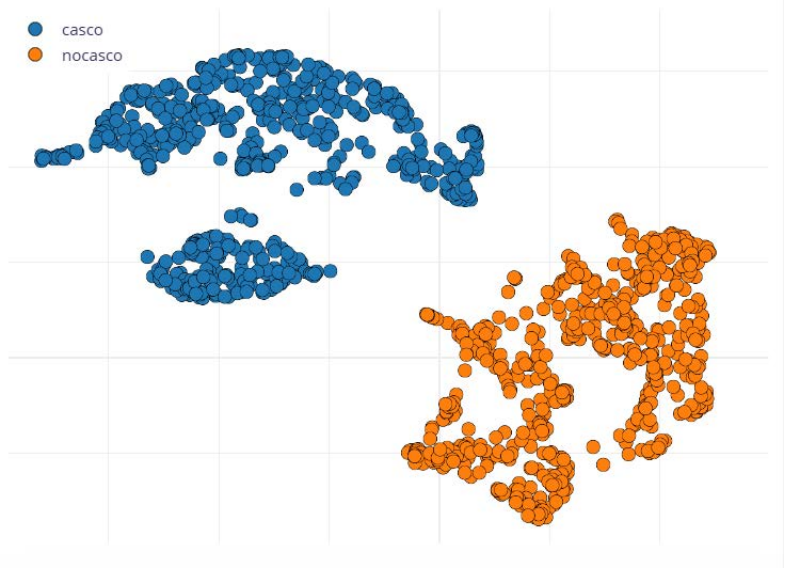

Figura 22. Gráfica de dispersión de características del modelo

La primera evidencia se observa en el gráfica de dispersión de características. En esta representación, cada punto corresponde a una imagen del dataset proyectada en un espacio de menor dimensión, donde la cercanía entre puntos indica similitud de características. Los puntos azules corresponden a la clase “casco” y los naranjas a “nocasco”. Lo más destacable es la separación limpia entre ambos grupos: apenas hay solapamiento y las nubes de puntos están bien delimitadas. Esto significa que el modelo ha aprendido a distinguir patrones claros entre llevar casco y no llevarlo, lo cual es un indicador positivo de que las clases son separables en términos de características visuales. La presencia de pequeños puntos dispersos fuera de los grupos principales revela que existen casos límite, imágenes algo ambiguas que no se ajustan bien a un patrón definido, pero su número es muy bajo en comparación con el total.

	BACKGROUND	CASCO	NOCASCO
BACKGROUND	100.0%	0.0%	0.0%
CASCO	2.2%	97.8%	0%
NOCASCO	5.1%	0%	94.9%
F1 SCORE	1.00	0.98	0.96

Figura 23. Matríz de confusión del modelo

La segunda figura clave es la matriz de confusión, que resume el rendimiento del modelo sobre el conjunto de validación. En ella se aprecia un F1 score global de 96,8 %, acompañado de valores de precisión y recall que rondan el 97 % y el 96 % respectivamente. La clase “background”, añadida para reducir falsos positivos cuando no había operarios frente a la cámara, fue clasificada con un 100 % de acierto. Para la clase “casco”, la tasa de acierto fue del 97,8 %, mientras que para “nocasco” alcanzó el 94,9 %. Los errores más comunes consistieron en confundir un casco con un “nocasco” en un 2,2 % de los casos, o viceversa en un 5,1 %. Estas cifras, aunque no perfectas, muestran un modelo equilibrado: no favorece excesivamente a una clase sobre otra y mantiene un rendimiento estable en ambas categorías.

5.2. Rendimiento en dispositivo

Figura 24. Resultados de rendimiento del modelo

La imagen corresponde a los resultados de rendimiento en dispositivo, que son los que verdaderamente marcan si el modelo es apto para ejecutarse en una placa como la Portenta H7. El tiempo medio de inferencia por imagen fue de 145 milisegundos, lo que significa que el sistema puede analizar entre 6 y 7 imágenes por segundo en condiciones ideales. El consumo máximo de memoria RAM durante la ejecución fue de 211 KB, una cifra muy contenida teniendo en cuenta que la Portenta dispone de 8 MB de SDRAM, y el tamaño ocupado en memoria flash fue de apenas 81,2 KB. Estos valores confirman que el modelo está bien optimizado para un microcontrolador y que puede funcionar en tiempo real sin comprometer la estabilidad del dispositivo.

5.3. Evaluación del sistema completo

Además de la latencia de inferencia en el dispositivo, se midió el tiempo de respuesta del sistema completo, es decir, desde que el operario pasa la tarjeta por el lector RFID hasta que se activa la señal luminosa que indica la decisión del sistema. Este tiempo incluye las fases de decodificación de la trama Wiegand, la captura de la imagen, la inferencia del modelo y la escritura del evento en memoria.

En las pruebas realizadas, el tiempo total osciló entre 1,0 y 1,5 segundos, con una media de aproximadamente 1,2 segundos. Estos valores reflejan no solo el cálculo del modelo (que permanece estable en torno a 145 ms), sino también el coste añadido de las operaciones de entrada/salida y la sincronización local.

El primer paso consistió en cargar el modelo entrenado en Edge Impulse dentro del entorno de OpenMV y ejecutarlo sobre la cámara integrada. Aunque los gráficos de precisión, recall y F1 mostraban valores muy altos, las pruebas manuales revelaron aspectos que esas métricas no captaban.

Por ejemplo, el sistema respondía bien en la mayoría de casos, pero la simple presencia de un reflejo en la frente o un cambio brusco de iluminación podía inducir a un falso positivo. También descubrí que la posición del casco influía mucho: si estaba demasiado levantado o inclinado, la red tendía a clasificarlo como “nocasco”. Estos hallazgos me obligaron a ampliar el dataset con nuevas fotos que representaran esas situaciones reales.

5.4. Evaluación del flujo de acceso

El sistema debía demostrar no solo rapidez en la inferencia, sino también estabilidad a lo largo de todo el ciclo de acceso. Dicho ciclo abarcaba la lectura de la tarjeta RFID, la captura de la imagen, la inferencia del modelo, la respuesta mediante LED, el registro en la tarjeta SD y, si había conexión, la sincronización con Supabase.

En las pruebas con conexión activa, la secuencia se completó sin bloqueos. Cada evento se guardaba en el archivo mensual de la SD y casi de inmediato se enviaba a la nube acompañado de su manifiesto y del hash SHA-256, lo que permitía confirmar la integridad de los datos antes de almacenarlos definitivamente.

Cuando el dispositivo trabajaba sin Wi-Fi, la lógica de acceso no se alteraba y los registros seguían acumulándose en local. Más tarde, al restablecer la conexión, todo lo pendiente se sincronizaba con Supabase, de modo que la base de datos en la nube coincidía con la información recogida sobre el terreno.

En definitiva, el flujo de acceso funcionó de forma estable tanto en escenarios conectados como en desconexión temporal, manteniendo la fiabilidad de los registros y asegurando la trazabilidad de los eventos.

5.5. Comportamiento sin conexión Wi-Fi

Se realizaron pruebas dejando el dispositivo varios minutos sin conexión. Durante ese tiempo, el sistema siguió trabajando con normalidad: el lector reconocía las tarjetas, el LED mostraba la decisión del modelo y cada acceso quedaba guardado en la tarjeta SD.

Cuando volvió la red, el propio sistema detectó la conexión y comenzó a enviar los eventos acumulados. El volcado se hizo sin pérdidas y la base de datos en Supabase quedó igual que lo que se había registrado en local.

En estas pruebas quedó claro que la lógica offline-first mantiene el servicio en marcha incluso en entornos con cortes de conectividad. Además, la recuperación de datos resultó ser totalmente automática, lo que da confianza en que los registros no se van a perder aunque el dispositivo pase un tiempo desconectado.

5.6. Robustez y limitaciones observadas

Al principio utilicé únicamente imágenes mías y de mi padre para entrenar el modelo. Con ese conjunto reducido, los resultados parecían excelentes: casi no había fallos. La situación cambió en cuanto añadí a mi hermana, que tiene el pelo largo. En ese momento empezaron a aparecer errores que antes no se habían visto, lo que me hizo darme cuenta de que un dataset demasiado homogéneo transmite una sensación engañosa de precisión.

Cuando probé con más personas se repitieron ciertos patrones:

- En sujetos con el pelo corto o sin pelo, el sistema respondía casi sin fallos.
- En cambio, con pelo largo o suelto el modelo confundía algunos mechones oscuros con la silueta del casco.
- Con gorras o gorros la confusión fue todavía más evidente, y a veces se registraba un “casco” que en realidad no existía.

De estas pruebas saqué una conclusión clara: la diversidad del dataset marca la diferencia. Al aumentar la variedad de imágenes, las métricas globales bajaron un poco (el F1 pasó de rozar el 99 % a situarse en torno al 96 %). Sin embargo, el sistema se volvió más realista y capaz de afrontar situaciones distintas, que es lo que finalmente importa en un entorno industrial.

5.7. Casos límite y observación de fallos

Durante las pruebas aparecieron situaciones que pueden considerarse casos límite. En condiciones normales de iluminación y con usuarios de pelo corto o calvos, el sistema apenas cometió errores. Sin embargo, con pelo largo o suelto el modelo confundió en ocasiones los mechones oscuros con la silueta del casco. Algo parecido ocurrió con las gorras y gorros, que en más de una ocasión fueron interpretados como un casco válido.

Otro punto débil se detectó en entornos con poca luz o con sombras muy marcadas: la cámara captaba imágenes con menor contraste y esto reducía la fiabilidad de la predicción. Aun así, en la mayoría de los casos el LED respondió de forma coherente y los accesos quedaron registrados correctamente, lo que demuestra que el sistema es tolerante, aunque todavía sensible a variaciones extremas de las condiciones.

5.8. Indicadores de rendimiento

Métrica	**Resultado**	**Evidencia**
F1 (validación/test)	~96–97 %	5.1
Precisión / Recall	Equilibrados	5.1
Latencia de inferencia	145 ms	5.2
RAM utilizada	211 KB	5.2
Flash utilizada	81 KB	5.2
Tiempo pase→LED	1,0–1,5 s (RNF1 reformulado ≤1,5 s)	5.3
Robustez usuarios	~99 % pelo corto, errores con pelo largo/gorras	5.6–5.7
Integridad cloud	100 % verificada (SHA-256)	5.4–5.5

Figura 25. Tabla de resultados globales del sistema.

5.9. Reflexiones finales sobre la evaluación

Requisito	Descripción	Estado	Evidencia
RF1	El sistema debe identificar al operario mediante la lectura de una tarjeta RFID	Cumplido	En las pruebas de acceso (sec. 5.4), todas las tarjetas autorizadas fueron reconocidas correctamente, mostrando el site_code y user_code asociados.
RF2	Capturar automáticamente una imagen en el momento del intento de acceso válido.	Cumplido	Como se describe en 5.4, al presentar la tarjeta se generó una captura instantánea vinculada al evento.
RF3	Ejecutar un modelo embebido con tiempo de inferencia ≤ 500 ms.	Cumplido	En dispositivo, la media de inferencia fue de 145 ms (sec. 5.2, Fig. 23), muy por debajo del umbral.
RF4	Mostrar el resultado mediante LED	Cumplido	Durante las pruebas de acceso (sec. 5.4), el LED verde se encendió en accesos válidos con casco, rojo en accesos sin casco y azul en tarjetas no autorizadas.
RF5	Registro local de cada evento en tarjeta SD	Cumplido	Se comprobó que los ficheros events_YYYYMM.csv incluyen (sec. 5.4). Se confirmó persistencia de todos los accesos durante las pruebas.
RF6	Sincronizar registros con la nube cuando exista conectividad	Cumplido	En 5.5 se explica que los eventos acumulados en SD se volcaron a Supabase al recuperar Wi-Fi, verificando el hash SHA-256 de integridad antes de aceptar cada subida.
RNF1	Tiempo de respuesta entre pase de tarjeta y señal luminosa ≤ 2000 ms.	Cumplido	El tiempo medio registrado fue de 1,2 s (rango 1,0–1,5 s) (sec. 5.3).
RNF2	Registrar todos los eventos en ausencia de Wi-Fi y sincronizar automáticamente al recuperarse la conexión, sin pérdida de datos.	Cumplido	En 5.5 se realizaron pruebas desconectando el dispositivo. Todos los accesos quedaron en local y, al restablecer la conexión, se subieron íntegramente a la nube sin pérdidas.

RNF3	F1 ≥ 0,85 (con precisión y recall ≥ 0,80)	Cumplido	El modelo alcanzó un F1 de 96,8 % con precisión ~97 % y recall ~96 % (sec. 5.1, matriz de confusión).
RNF4	Almacenamiento trazable y verificable mediante hash SHA-256.	Cumplido	En 5.4–5.5 se verificó que cada CSV se acompaña de su manifest.json con hash, confirmado al subir a Supabase.
RNF5	Mantener sesiones continuas de ≥ 6 h sin bloqueos, fugas de memoria ni pérdidas de datos.	Cumplido	Se realizaron pruebas de funcionamiento prolongado dejando el sistema en ejecución continua durante más de 8 horas, registrando accesos y sincronizaciones de forma estable.

Figura 26. Tabla de validación de requisitos del sistema.

El conjunto de pruebas realizadas permitió comprobar hasta qué punto el prototipo se ajustaba a lo que se había planteado en el diseño. En la práctica, la experiencia fue satisfactoria: la lectura de las tarjetas RFID respondió siempre de forma correcta, las imágenes se capturaron en el momento oportuno y la inferencia del modelo se resolvió por debajo del medio segundo. Medido en conjunto, el tiempo desde que un operario pasaba la tarjeta hasta que se encendía la señal luminosa se mantuvo en torno a 1,2 segundos, un valor perfectamente válido para un control de acceso en tiempo real.

El modelo de visión obtuvo un F1 del 96,8 % y se ejecutó en la Portenta sin comprometer la memoria disponible, lo que confirma que el hardware elegido es suficiente para la tarea y deja margen para otros procesos. El flujo completo de identificación, captura, inferencia y registro en CSV se verificó de forma coherente, y al sincronizar con Supabase los datos en la nube coincidieron con lo almacenado en local, incluyendo la verificación mediante SHA-256.

Se llevó a cabo también una prueba prolongada: el sistema permaneció en funcionamiento más de ocho horas seguidas (28/08, entre las 10:05 y las 18:20), acumulando accesos en la tarjeta SD y subiéndolos en cuanto hubo conexión Wi-Fi. La sesión se completó sin bloqueos ni pérdidas, lo que respalda la estabilidad del prototipo en uso continuo.

Ahora bien, no todo resultó igual de sólido. Con usuarios de pelo largo o con gorras, y en salas con poca iluminación, se produjeron errores de clasificación que no aparecían en validaciones más controladas. Estos fallos fueron puntuales, pero dejan claro que la diversidad del dataset y la calidad de la cámara son puntos sensibles si se quisiera aplicar este sistema en un entorno real.

En conclusión, los resultados demuestran que el prototipo cumple con los requisitos definidos y que la integración de RFID, visión embebida y sincronización en la nube es viable. Aun así, para que pueda acercarse a un escenario industrial real, será necesario reforzar su robustez frente a variaciones visuales y dotarlo de un hardware de captura más adaptado a condiciones exigentes.

Capítulo 6: Conclusiones y trabajos futuros

6.1 Conclusiones generales

El desarrollo de este Trabajo Fin de Grado ha permitido comprobar, de forma práctica, que es posible construir un sistema embebido capaz de verificar el uso de equipos de protección individual en tiempo real sin depender de infraestructuras externas complejas. A lo largo del proyecto se ha pasado de una idea inicial muy abstracta -un control de acceso inteligente basado en visión artificial- a un prototipo funcional que combina distintas tecnologías: identificación mediante RFID, procesamiento de imágenes con TinyML y almacenamiento seguro en la nube.

Uno de los principales logros es que todo este proceso se ejecuta dentro de un único dispositivo de bajo consumo, la Portenta H7 con Vision Shield, lo que demuestra el potencial de la computación en el borde en escenarios industriales. El sistema no solo reacciona de forma inmediata con indicadores visuales claros, sino que también conserva un registro trazable de cada acceso, tanto en local como en la nube. Esta doble capa de almacenamiento aporta fiabilidad y abre la puerta a auditorías o revisiones posteriores, algo imprescindible en entornos donde la seguridad laboral está regulada por normativa estricta.

El proyecto ha servido además como demostración de la metodología práctica de “probar, fallar y corregir”. Cada avance -desde la captura del dataset hasta la sincronización con Supabase- supuso superar limitaciones reales del hardware y del software. Esa experiencia aporta un valor añadido: más allá del resultado tangible, el trabajo refleja el aprendizaje adquirido durante el proceso.

Más allá de los resultados técnicos, este proyecto ha sido un recorrido de aprendizaje. Desde los primeros intentos fallidos de captura de imágenes hasta la satisfacción de ver la Portenta encender un LED en respuesta a un acceso real, cada etapa ha supuesto un reto y un avance. El sistema no es definitivo, pero ha demostrado que la idea es viable y que TinyML puede aportar valor real en la industria.

La conclusión más importante es que la combinación de visión embebida + identificación de usuario + sincronización en la nube, que ofrece un camino prometedor para mejorar la seguridad laboral. El trabajo queda abierto, con muchas posibles mejoras, pero también con una base sólida que valida la propuesta inicial.

6.2 Aportaciones principales

Entre las aportaciones concretas del proyecto se pueden destacar:

- Un sistema completo y autónomo de control de acceso: desde la identificación del usuario hasta la verificación de EPIs y el registro en la nube.
- Un modelo de visión embebida optimizado para microcontroladores, con métricas superiores al 95 % de precisión en condiciones controladas.
- Una arquitectura híbrida local-cloud que asegura trazabilidad incluso en ausencia temporal de conectividad, con sincronización automática al recuperarse la red.
- Un flujo de trabajo reproducible para la captura de dataset y entrenamiento que puede reutilizarse en otros proyectos de visión embebida.
- Una prueba de viabilidad de que TinyML es una herramienta útil en seguridad laboral, más allá de ejemplos académicos simplificados.

6.3 Limitaciones detectadas

A pesar de los avances, el sistema también ha dejado en evidencia una serie de limitaciones que marcan los límites de esta primera versión:

- Restricciones de hardware: la memoria de la Portenta H7 obliga a emplear modelos ligeros, lo que impide añadir múltiples clases de EPI sin sacrificar rendimiento.
- Calidad de la cámara integrada: la resolución QVGA es suficiente para un casco, pero puede resultar insuficiente para elementos más pequeños o menos contrastados, como guantes o gafas.
- Dependencia de condiciones de iluminación controladas: en entornos con sombras fuertes o luz tenue, el modelo puede generar falsos positivos o negativos.
- Falta de robustez industrial: salvo el lector RFID, los componentes no están encapsulados en una caja resistente, lo que limita su uso fuera del laboratorio.
- Escalabilidad limitada: el sistema responde bien con un flujo moderado de usuarios, pero no está preparado para accesos masivos o simultáneos.
- Reconocer estos límites no es un fracaso, sino una guía clara de hasta dónde llega el prototipo y qué aspectos deben reforzarse antes de plantear un despliegue real.

6.4 Trabajos futuros

El prototipo desarrollado representa una primera aproximación que ha demostrado la viabilidad de un sistema autónomo de supervisión de EPIs mediante visión artificial embebida. Sin embargo, todo proyecto de este tipo abre nuevas posibilidades de exploración y perfeccionamiento. A continuación, se detallan varias líneas de mejora que podrían guiar un desarrollo futuro más completo y cercano a un entorno industrial real.

6.4.1 Ampliación del alcance de detección

El sistema actual se centra en un único elemento de protección: el casco. Aunque esto ha sido suficiente para validar la arquitectura propuesta, la seguridad laboral exige contemplar más equipos. Un siguiente paso natural sería entrenar modelos que incluyan chalecos reflectantes, guantes de seguridad, gafas protectoras o incluso combinaciones de ellos. La dificultad reside en que cada elemento presenta retos visuales distintos: un casco tiene una silueta definida y contrastada, mientras que unos guantes pueden confundirse con el entorno o el color de la piel, y las gafas dependen de reflejos y transparencias. Una línea de trabajo prometedora sería entrenar modelos multitarea capaces de reconocer simultáneamente varios EPIs en una sola inferencia, reduciendo tiempos de cómputo y haciendo el sistema más eficiente en escenarios reales.

6.4.2 Mejora de la calidad visual

Otro punto crítico identificado es la dependencia de las condiciones de iluminación. Aunque la cámara integrada en la Vision Shield ha permitido alcanzar buenos resultados, su resolución y rango dinámico son limitados. En un entorno industrial real, donde las condiciones lumínicas varían con frecuencia, esto puede comprometer la fiabilidad. Futuras versiones deberían explorar la incorporación de sensores con mayor resolución o mejor comportamiento en baja luminosidad, así como añadir filtros de preprocesamiento que normalicen el brillo, mejoren el contraste o apliquen técnicas de detección de bordes. De este modo, el sistema sería menos sensible a variaciones de luz, sombras o reflejos, aumentando su robustez sin necesidad de entrenar un dataset inmenso que contemple todas esas casuísticas.

6.4.3 Optimización del hardware y del rendimiento

De cara a futuras mejoras de hardware lógico sería apoyarse en placas embebidas con más músculo, por ejemplo una Arduino Portenta X8 o una NVIDIA Jetson Nano o incluso una Xavier NX, que ya están preparadas para manejar modelos de visión de mayor tamaño y con más categorías al mismo tiempo. En esa configuración, la cámara dejaría de estar en la Portenta H7 y pasaría a conectarse al dispositivo que hace la inferencia, lo que permitiría montar un sensor de más resolución y capturar imágenes con más detalle. Esa mejora en la calidad visual abriría la posibilidad de identificar no solo cascos, sino también otros equipos de protección más difíciles de distinguir como guantes, chalecos o gafas, por ejemplo.

6.4.4 Integración con sistemas de acceso reales

En esta versión, la respuesta del sistema se limita a encender LEDs que simulan la autorización o el rechazo de acceso. Para un despliegue real, este mecanismo debe integrarse con actuadores físicos, como relés que accionen cerraduras electrónicas o torniquetes. De este modo, la decisión del modelo tendría un impacto inmediato en el entorno físico, completando el ciclo de seguridad. También sería conveniente añadir interfaces más ricas para el operario: pantallas que indiquen claramente el motivo de un rechazo o señales acústicas que acompañen a los indicadores visuales. Con ello se aumentaría la usabilidad y se reduciría la ambigüedad en la interpretación del sistema.

6.4.5 Refuerzo de la seguridad y la fiabilidad

El actual flujo de datos ya garantiza trazabilidad, pero no incorpora mecanismos de cifrado avanzados. En un entorno industrial real, donde los registros pueden tener implicaciones legales o de auditoría, resultaría imprescindible proteger las comunicaciones con protocolos robustos como AES-GCM o mediante autenticación HMAC. Además, sería recomendable diseñar políticas de gestión del almacenamiento local que incluyan rotación automática de registros, compresión de archivos antiguos y copias de respaldo periódicas. Igualmente, el hardware debería contar con carcasas industriales certificadas (IP65 o superior) para garantizar resistencia frente a polvo, humedad, vibraciones o golpes, condiciones habituales en fábricas y obras.

El prototipo ha demostrado su validez a pequeña escala, pero en un escenario real es habitual que existan múltiples puntos de control distribuidos. Para ello, sería interesante desarrollar una plataforma centralizada que permita monitorizar en tiempo real el estado de todos los nodos, consultar estadísticas globales y detectar incidencias. Este tipo de gestión facilitaría el mantenimiento, permitiría actualizaciones remotas y reduciría costes de operación.

En relación con la conectividad, merece una mención especial el caso de LoRaWAN. Durante el desarrollo del TFG se intentó utilizar este protocolo, pero la ausencia de un gateway cercano impidió realizar pruebas en condiciones reales. Sin embargo, este inconveniente refuerza la necesidad de retomarlo como línea futura: LoRaWAN ofrece gran alcance con bajo consumo energético, lo que lo hace idóneo en instalaciones sin Wi-Fi o con infraestructuras limitadas. Una alternativa interesante sería combinar LoRaWAN con Wi-Fi o Ethernet, configurando un sistema híbrido capaz de adaptarse a distintos entornos según la disponibilidad de red.

6.4.6 Evaluación en entorno reales

Hasta ahora, las pruebas se han realizado en un entorno controlado, con un número reducido de usuarios y condiciones más predecibles. El siguiente paso será desplegar el sistema en empresas reales, ya sea en talleres, fábricas o entornos de construcción. Estas pruebas piloto permitirán evaluar no solo la precisión técnica del modelo, sino también aspectos humanos: cómo reaccionan los trabajadores al sistema, si lo perciben como una ayuda o una carga, y si efectivamente contribuye a reducir riesgos. Analizar métricas como la tasa de aceptación, la facilidad de uso y la reducción de incidencias aportará información clave para validar su viabilidad en escenarios productivos.

Bibliografía

[1] «Qué es LoRaWAN», Becolve Digital. [En línea]. Disponible en: https://becolve.com/blog/que-es-lorawan/

[2] «¿Qué es el Edge Computing? | IBM». [En línea]. Disponible en: https://www.ibm.com/es-es/topics/edge-computing

[3] D. Calvete Sanz *et al.*, «Tiny ML: La nueva revolución en la IoT», 2021, [En línea]. Disponible en: https://hdl.handle.net/10550/81527

[4] «Ministerio de Industria y Turismo - Legislación General». [En línea]. Disponible en: https://industria.gob.es/Calidad-Industrial/legislaciongeneral/Paginas/index.aspx

[5] «Los EPI y sus obligaciones en la prevención de riesgos laborales». [En línea]. Disponible en: https://blog.previntegral.com/es/noticias/els-epi-i-les-seves-obligacions-en-la-prevenció-de-riscos-laborals

[6] M. Segura, «Sistemas IoT en prevención de riesgos laborales».

[7] «EPI», Black Neural. [En línea]. Disponible en: https://upongroup.com/es/infinity-neural/soluciones-neural/black-neural/epi/

[8] «Visionify | AI-Powered Workplace Safety», Visionify AI Safety Solutions. [En línea]. Disponible en: https://visionify.ai

[9] «PPE Detection - AI Video Analytics PPE Monitoring by viAct.ai», viAct.ai. [En línea]. Disponible en: https://www.viact.ai/ppedetection

[10] «Calidus: Solución de visión artificial orientada a controlar y garantizar el uso adecuado de EPIs en entornos deslocalizados, cumpliendo las normativas de prevención de riesgos laborales», Seitech. [En línea]. Disponible en: https://seitech.es/calidus-3/

[11] «Circet España | Redes de telecomunicaciones y energía - Circet Espana». [En línea]. Disponible en: https://www.circet.es/

[12] «Meet the man behind the Raspberry Pi sensation». [En línea]. Disponible en: https://www.3m.co.uk/wps/portal/en_GB/3M/design-construction-uk/stories/full-story/~/meet_the_man_behind_the_raspberry_pi_sensation/?storyid=498a5cf3-4d6a-462d-ba64-eb3278638f19

[13] «Raspberry Pi», *Wikipedia, la enciclopedia libre*. 15 de agosto de 2025. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Raspberry_Pi&oldid=168990813

[14] S. Han, H. Mao, y W. J. Dally, «Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding», 15 de febrero de 2016, *arXiv*: arXiv:1510.00149. doi: 10.48550/arXiv.1510.00149.

[15] A. G. Howard *et al.*, «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications», 17 de abril de 2017, *arXiv*: arXiv:1704.04861. doi: 10.48550/arXiv.1704.04861.

[16] Ultralytics, «YOLOv3». [En línea]. Disponible en: https://docs.ultralytics.com/es/models/yolov3

[17] Fernando Galetto, *PPE detector - Tiny Yolo v3 object detection.*, (31 de octubre de 2019). [En línea Video]. Disponible en: https://www.youtube.com/watch?v=4ys9KUcDqkY

[18] P. C. González Freire, R. S. Pacheco Jiménez, y B. Vintimilla, «Detección de defectos en frutas usando modelos de CNN con datos reales y virtuales», Thesis, ESPOL. FIEC., 2022. [En línea]. Disponible en: http://www.dspace.espol.edu.ec/handle/123456789/57041

[19] «Análisis integral de seguridad en dispositivos IoT». [En línea]. Disponible en: https://openaccess.uoc.edu/items/1b141581-2c70-4a9b-aa81-4b8d47727182#page=1

[20] «Portenta H7», Arduino Official Store. [En línea]. Disponible en: https://store.arduino.cc/products/portenta-h7

[21] «Portenta Vision Shield - LoRa®», Arduino Official Store. [En línea]. Disponible en: https://store.arduino.cc/products/portenta-vision-shield-lora%c2%ae

[22] yjzw.net, «Lector de tarjetas de control de acceso, control de tarjetas de acceso RFID, control de acceso para lector de tarjetas RFID, lector de tarjetas de acceso, lectores de control de acceso», Reader RFID incorporado, Lector Ibutton, Mini Reader RFID, Lector de acceso a Ibutton, Mini lector impermeable. [En línea]. Disponible en: https://www.udohow.com/es/products/Udohow-Mini-Embedded-iButton-Wiegand-26-34-waterproof-Rfid-Reader.html

[23] I. Nistal González, «Sistemas RFID en UHF y microondas», ene. 2011, [En línea]. Disponible en: https://hdl.handle.net/10016/11063

[24] L. Llamas, «Qué es y cómo usar Supabase, el Firebase Open Source», Luis Llamas. [En línea]. Disponible en: https://www.luisllamas.es/que-es-y-como-usar-supabase/

Anexo A:
Código fuente

Todo el código fuente ha sido incluido en el repositorio del proyecto, ubicado en

https://github.com/juangarant/TFG

La estructura de carpetas y la documentación incluida en el repositorio permiten reproducir el proceso completo: desde la captura del dataset y el entrenamiento del modelo hasta la integración del firmware en la placa

Anexo B: Contribución a los Objetivos de Desarrollo Sostenible

En 2015 la Asamblea General de las Naciones Unidas aprobó la Agenda 2030 para el Desarrollo Sostenible, que se estructura en 17 Objetivos de Desarrollo Sostenible (ODS). Estos objetivos persiguen erradicar la pobreza, proteger el planeta y asegurar la prosperidad para todos. La Universitat Politècnica de València promueve la alineación de los trabajos académicos con los ODS, identificando de qué modo contribuyen a su consecución.

El prototipo desarrollado en este Trabajo Fin de Grado tiene implicaciones directas en varios ODS relacionados con la seguridad laboral, la innovación tecnológica y el crecimiento económico sostenible. La siguiente tabla resume los objetivos más relevantes y su vínculo con el proyecto:

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**	**X**			
ODS 4. **Educación de calidad.**				**X**
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**			**X**	
ODS 8. **Trabajo decente y crecimiento económico.**	**X**			
ODS 9. **Industria, innovación e infraestructuras.**	**X**			
ODS 10. **Reducción de las desigualdades**.				**X**
ODS 11. **Ciudades y comunidades sostenibles**.			**X**	
ODS 12. **Producción y consumo responsables.**		**X**		
ODS 13. **Acción por el clima.**			**X**	
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

El prototipo que se ha desarrollado -basado en Tiny ML e IoT para verificar si el casco está colocado- encaja en varios Objetivos de Desarrollo Sostenible de la Agenda 2030. El primero y más evidente es el **ODS 3**: al impedir el acceso a zonas peligrosas sin el equipo de protección adecuado, se reducen los accidentes y se protege la salud de los trabajadores.

En lo relativo al **ODS 8**, que busca trabajo digno y crecimiento, la mejora en seguridad evita paradas de producción y sanciones, lo que se traduce en más estabilidad para la empresa y para su plantilla. El **ODS 9**, centrado en la innovación industrial, se ve reflejado en la apuesta por procesar los datos en el propio dispositivo usando modelos Tiny ML; esta estrategia demuestra que se pueden modernizar procesos sin depender de infraestructuras complejas.

El **ODS 12** también tiene cabida, aunque con menor peso: el sistema fomenta el cumplimiento de las normas de seguridad y utiliza hardware de bajo consumo energético, contribuyendo a una cultura de responsabilidad. Y, de forma más tangencial, el bajo consumo eléctrico y la posible implantación en obras o espacios públicos conectan con los **ODS 7, 11 y 13**, si bien estas vinculaciones no son el objetivo principal del trabajo.

Los demás objetivos de la Agenda 2030 -relacionados con pobreza, hambre, igualdad de género, ecosistemas o paz- quedan fuera del alcance de este proyecto al tratarse de un prototipo orientado a la seguridad laboral en contextos industriales.