PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

ESCUELA HÁBITAT, INFRAESTRUCTURA Y CREATIVIDAD

TRABAJO DE INTEGRACIÓN CURRICULAR PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

PROTOTIPO DE IOT BASADO EN TINYML EN ESP32-S3 PARA CONTEO AUTOMÁTICO DE PASAJEROS EN UNA PARADA DE TRANSPORTE RURAL

ADRIÁN DANILO CARRILLO ZAMBRANO

TUTOR:

LAURA ROSA GUERRA TORREALBA

IBARRA – ECUADOR

JULIO, 2026

Ibarra, julio de 2026

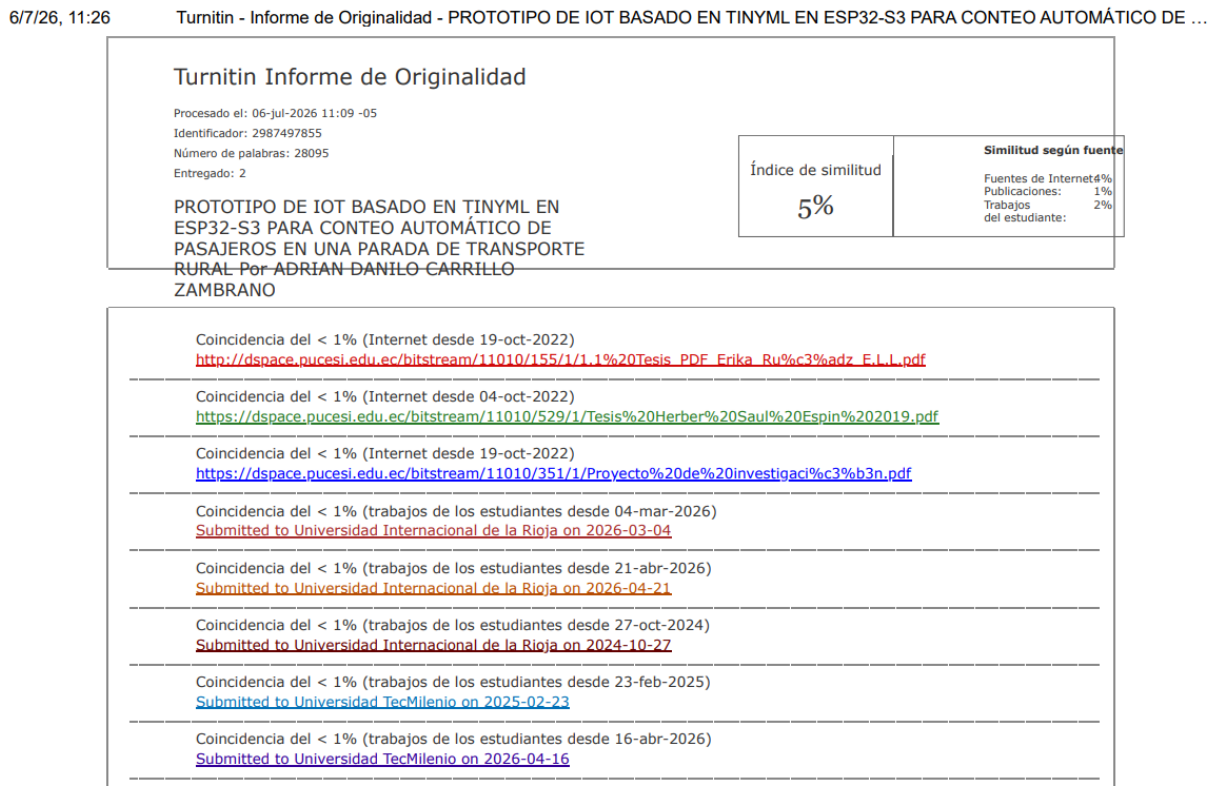
CERTIFICACIÓN TUTOR

En mi calidad de Tutor del Trabajo de titulación titulado:

PROTOTIPO DE IOT BASADO EN TINYML EN ESP32-S3 PARA CONTEO AUTOMÁTICO DE PASAJEROS EN UNA PARADA DE TRANSPORTE RURAL, presentado por el estudiante ADRIÁN DANILO CARRILLO ZAMBRANO con cédula de ciudadanía N° 0401965306, para obtener el Título de INGENIERÍA EN TECNOLOGIAS DE LA INFORMACIÓN.

Certifico que el trabajo cumple con todos los parámetros establecidos, mediante el cual el estudiante demuestra el desarrollo de competencias en el campo de conocimiento de su profesión con un nivel de argumentación coherente, para ser sometido a la evaluación por parte de los lectores.

Adicionalmente, se adjunta el certificado de porcentaje de originalidad de TURNITIN.


6/7/26, 11:26 Turnitin - Informe de Originalidad - PROTOTIPO DE IOT BASADO EN TINYML EN ESP32-S3 PARA CONTEO AUTOMÁTICO DE ...

Turnitin Informe de Originalidad

Procesado el: 06-jul-2026 11:09 -05
Identificador: 2987497855
Número de palabras: 28095
Entregado: 2

PROTOTIPO DE IOT BASADO EN TINYML EN ESP32-S3 PARA CONTEO AUTOMÁTICO DE PASAJEROS EN UNA PARADA DE TRANSPORTE RURAL Por ADRIAN DANILO CARRILLO ZAMBRANO

Índice de similitud	Similitud según fuente
5%	Fuentes de Internet 4% Publicaciones: 1% Trabajos del estudiante: 2%

Coincidencia del < 1% (Internet desde 19-oct-2022)
http://dspace.pucesi.edu.ec/bitstream/11010/155/1/1.1%20Tesis_PDF_Erika_Ru%c3%adz_E.L.L.pdf

Coincidencia del < 1% (Internet desde 04-oct-2022)
https://dspace.pucesi.edu.ec/bitstream/11010/529/1/Tesis%20Herber%20Saul%20Espin%202019.pdf

Coincidencia del < 1% (Internet desde 19-oct-2022)
https://dspace.pucesi.edu.ec/bitstream/11010/351/1/Proyecto%20de%20investigaci%c3%b3n.pdf

Coincidencia del < 1% (trabajos de los estudiantes desde 04-mar-2026)
Submitted to Universidad Internacional de la Rioja on 2026-03-04

Coincidencia del < 1% (trabajos de los estudiantes desde 21-abr-2026)
Submitted to Universidad Internacional de la Rioja on 2026-04-21

Coincidencia del < 1% (trabajos de los estudiantes desde 27-oct-2024)
Submitted to Universidad Internacional de la Rioja on 2024-10-27

Coincidencia del < 1% (trabajos de los estudiantes desde 23-feb-2025)
Submitted to Universidad TecMilenio on 2025-02-23

Coincidencia del < 1% (trabajos de los estudiantes desde 16-abr-2026)
Submitted to Universidad TecMilenio on 2026-04-16

(f):_____ Laura Guerra Torrealba
Firmado digitalmente por Laura Guerra Torrealba
Fecha: 2026.07.23 16:10:45 -05'00'

PhD. Guerra Torrealba Laura Rosa
TUTOR DE TRABAJO
C.C.: 1757842784

PÁGINA DE APROBACIÓN DEL TRIBUNAL

El tribunal examinador, aprueba el presente trabajo en nombre de la Pontificia Universidad Católica del Ecuador Ibarra:

(f): ……

Laura Guerra Torrealba

Firmado digitalmente por Laura Guerra Torrealba
Fecha: 2026.07.23 16:11:29 -05'00'

PhD. Laura Rosa Guerra Torrealba

C.C.: 1757842784

Mgs. Stalin Arciniegas

Digitally signed by Mgs. Stalin Arciniegas
DN: cn=Mgs. Stalin Arciniegas gn=Mgs. Stalin Arciniegas c=EC Ecuador l=EC Ecuador o=PUCESI ou=Escuela de Ingeniería e=smarciniegas@pucesi.edu.ec
Reason: I am the author of this document
Location:
Date: 2026-07-23 15:53-05:00

(f):……

Msc. Stalin Marcelo Arciniegas Aguirre

C.C.: 1003496815

DARWIN MARCELO PILLO GUANOLUISA

Firmado digitalmente porDARWIN MARCELO PILLO GUANOLUISA
DN: cn=DARWIN MARCELO PILLO GUANOLUISA c=EC o=FIRMASEGURA S.A.S. ou=ENTIDAD DE CERTIFICACION DE INFORMACION e=darwin.marcelo.pillo@gmail.com
Motivo:Apruebo este documento
Ubicación:Ibarra
Fecha:2026-07-23 07:24-05:00

(f):……

Msc. Darwin Marcelo Pillo Guanoluisa

C.C.: 1003319660

ACTA DE CESIÓN DE DERECHOS

Yo, *ADRIÁN DANILO CARRILLO ZAMBRANO*, declaro conocer y aceptar la disposición del Art. 165 del Código Orgánico de Economía Social de los Conocimientos, Creatividad e Innovación, que manifiesta textualmente: "Se reconoce facultad de los autores y demás titulares de derechos de disponer de sus derechos o autorizar las utilizaciones de sus obras o prestaciones a título gratuito y oneroso, según las condiciones que determinen. Esta facultad podrá ejercerse mediante licencias libres, abiertas y otros modelos alternativos de licenciamiento o la renuncia".

Ibarra, 6 de julio de 2026

Adrian Danilo Carrillo Zambrano
Firmado digitalmente por Adrian Danilo Carrillo Zambrano
Fecha: 2026.07.23 16:16:24 -05'00'

(f):_____

ADRIÁN DANILO CARRILLO ZAMBRANO
C.C.: 0401965306

AUTORIA

Yo, *ADRIÁN DANILO CARRILLO ZAMBRANO*, portador de la cedula de ciudadanía N° 0401965306, declaro que el presente trabajo de investigación es de total responsabilidad de la autor@, y eximo expresamente a la Pontificia Universidad Católica del Ecuador Ibarra de posibles reclamos o acciones legales.

Adrian Danilo Carrillo Zambrano

Firmado digitalmente por Adrian Danilo Carrillo Zambrano
Fecha: 2026.07.23 16:16:37 -05'00'

(f):……..

ADRIÁN DANILO CARRILLO ZAMBRANO

C.C.: 0401965306

DEDICATORIA

A Dios, por ser mi guía constante y la fuente de mi fortaleza en cada etapa de este camino. Por darme la sabiduría para afrontar los desafíos, la serenidad para superar los momentos difíciles y la certeza de que nunca he caminado solo. Como nos recuerda su palabra: *"Mira que te mando que te esfuerces y seas valiente; no temas ni desmayes, porque Jehová tu Dios estará contigo en dondequiera que vayas"* (Josué 1:9), una promesa que ha sostenido cada uno de mis pasos.

A mis padres, Danilo Carrillo y Angélica Zambrano, a quienes dedico este logro con todo mi corazón. Gracias por su amor incondicional, por su esfuerzo incansable y por enseñarme, con su propio ejemplo, que la perseverancia, la honestidad y la dedicación son los cimientos sobre los que se construye cualquier sueño. Gracias por enseñarme que, con fe, para Dios ninguna meta es imposible, y por apoyarme sin descanso a lo largo de todos mis estudios. Cada una de mis metas alcanzadas lleva impreso su sacrificio, y todo lo que soy hoy se lo debo a ustedes.

A mis hermanos, Daniela y Matías, por ser mi compañía, mi alegría y mi motivación para seguir adelante. A pesar de nuestras diferencias y de que muchas veces la comunicación no fue la mejor, siempre traté de ser un ejemplo para ustedes y de demostrarles que, con esfuerzo, todo es posible. Ustedes me impulsan a dar siempre lo mejor de mí y a convertirme en alguien del que puedan sentirse orgullosos.

A mi tía Amparo Carrillo, quien ha sido una persona muy importante en mi vida. Gracias por enseñarme que con perseverancia y disciplina todo se puede lograr, y sobre todo por estar presente para mí y para mi familia en los momentos más difíciles. Su apoyo ha sido un pilar que jamás olvidaré.

A mis compañeros de clase, con quienes compartí esta etapa universitaria, sus desvelos, sus retos y también sus satisfacciones. Gracias por su amistad, por ser un grupo tan unido, por el apoyo mutuo y por las jornadas de estudio compartidas que convirtieron el día a día en un camino más llevadero y lleno de buenos recuerdos. Fueron un grupo que nunca olvidaré.

A todos ustedes dedico este trabajo, fruto de un esfuerzo que también es suyo.

AGRADECIMIENTOS

A Dios, por sobre todas las cosas, por regalarme la vida, la salud y la sabiduría necesarias para culminar esta importante etapa. Por sostenerme en los momentos de incertidumbre y por poner en mi camino a las personas correctas en el instante preciso.

A mi familia, por su apoyo incondicional, su paciencia infinita y por ser el soporte firme que nunca me faltó a lo largo de todos estos años de formación. Su confianza en mí fue, en muchas ocasiones, el motor que me permitió seguir adelante.

A la Pontificia Universidad Católica del Ecuador, Sede Ibarra, por abrirme sus puertas y brindarme una formación académica y humana de excelencia. En sus aulas no solo adquirí los conocimientos que hoy me permiten alcanzar esta meta, sino también valores y experiencias que me acompañarán a lo largo de toda mi vida profesional.

A mi tutora, la Dra. Laura Guerra, por su valiosa guía, su tiempo y su compromiso durante todo el desarrollo de este trabajo. Su acompañamiento, su criterio y sus oportunas observaciones fueron fundamentales para orientar este proyecto y llevarlo a buen término. Gracias por la dedicación y la exigencia con las que enriqueció cada etapa de esta investigación.

Finalmente, deseo expresar un agradecimiento muy especial al Msc. Paúl Enríquez, mi tutor durante las pasantías, quien, más que un jefe, se convirtió en un gran amigo. Gracias por su confianza, por sus consejos y por compartir conmigo, con generosidad, su experiencia tanto profesional como personal. Su ejemplo dejó en mí una huella que sin duda guiará mi camino en los años por venir.

A todas las personas que, de una u otra manera, formaron parte de este proceso, mi más sincero y profundo agradecimiento

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El servicio de transporte rural compartido de la ruta Bolívar - El Ángel, operado por la compañía TAXIBOCARCHI en la provincia del Carchi, Ecuador, funciona bajo un esquema que exige completar una cuota mínima de pasajeros antes de iniciar cada recorrido. La ausencia de un mecanismo que contabilice y notifique automáticamente el número de personas presentes en la parada genera tiempos de espera indeterminados para los pasajeros y una gestión ineficiente de las unidades, mientras que los sistemas de monitoreo convencionales resultan inviables por su alto costo e infraestructura compleja. Ante esta problemática, el presente trabajo tuvo como objetivo general desarrollar un prototipo de sistema IoT basado en Edge Computing y TinyML, mediante el microcontrolador XIAO ESP32-S3 Sense, para el conteo automático de pasajeros y la mejora de la coordinación operativa del servicio. Metodológicamente, el modelo de detección de objetos FOMO (Faster Objects, More Objects) se entrenó y se cuantizó a formato int8 en la plataforma Edge Impulse y se desplegó para ejecutar la inferencia localmente en el ESP32-S3. El nodo transmite únicamente el conteo numérico, nunca las imágenes, preservando así la privacidad de los usuarios, hacia un backend doble (Flask y NestJS) que persiste los eventos en Firebase Firestore, emite alertas mediante Telegram a los conductores y los expone en un panel web de monitoreo desarrollado bajo la metodología Scrum. La validación se ejecutó mediante 18 casos de prueba sobre los cuatro módulos del sistema. Los resultados muestran que el modelo alcanzó un F1 Score de 88.3 %, superando el umbral del 85 % establecido, con una huella de 69.4 KB de RAM y 68.9 KB de Flash y una latencia de inferencia de 314 ms. El flujo extremo a extremo, desde la captura hasta la visualización en el panel, se completó en 4.2 segundos, por debajo del criterio de 5 segundos; los 14 requisitos funcionales y los 18 casos de prueba se cumplieron en su totalidad, y la prueba de usabilidad arrojó un tiempo promedio de 87

segundos, inferior al máximo aceptable de 120 segundos. Se concluye que el prototipo cumplió los objetivos planteados y demuestra la viabilidad de aplicar TinyML y Edge Computing al conteo de pasajeros sobre hardware de bajo costo y de forma respetuosa con la privacidad. Se recomienda re-entrenar el modelo con imágenes capturadas en la propia ruta y bajo condiciones ambientales variables, ejecutar pruebas de robustez y avanzar hacia un despliegue piloto en condiciones reales de operación.

Palabras clave: TinyML; Edge Computing; visión artificial; conteo de pasajeros; ESP32-S3; FOMO; transporte rural; IoT.

ABSTRACT

The shared rural transport service on the Bolívar - El Ángel route, operated by the company TAXIBOCARCHI in the province of Carchi, Ecuador, works under a scheme that requires reaching a minimum quota of passengers before starting each trip. The absence of a mechanism to automatically count and report the number of people present at the stop produces indeterminate waiting times for passengers and inefficient management of the vehicles, while conventional monitoring systems are unfeasible due to their high cost and complex infrastructure. In response to this problem, the general objective of this work was to develop an IoT system prototype based on Edge Computing and TinyML, using the XIAO ESP32-S3 Sense microcontroller, for the automatic counting of passengers and the improvement of the service's operational coordination. Methodologically, the FOMO (Faster Objects, More Objects) object detection model was trained and quantized to int8 format on the Edge Impulse platform and deployed to run inference locally on the ESP32-S3. The node transmits only the numerical count, never the images, thus preserving user privacy, to a dual backend (Flask and NestJS) that persists the events in Firebase Firestore, sends alerts to drivers via Telegram, and displays them on a web monitoring dashboard developed under the Scrum methodology. Validation was carried out through 18 test cases across the four modules of the system. The results show that the model reached an F1 Score of 88.3%, exceeding the established 85% threshold, with a footprint of 69.4 KB of RAM and 68.9 KB of Flash and an inference latency of 314 ms. The end-to-end flow, from capture to display on the dashboard, was completed in 4.2 seconds, below the 5-second criterion; the 14 functional requirements and the 18 test cases were fully met, and the usability test yielded an average time of 87 seconds, below the maximum acceptable value of 120 seconds. It is concluded that the prototype met the proposed objectives and demonstrates the feasibility of

applying TinyML and Edge Computing to passenger counting on low-cost hardware in a privacy-respecting manner. It is recommended to retrain the model with images captured on the route itself and under variable environmental conditions, to carry out robustness tests, and to move toward a pilot deployment under real operating conditions.
Keywords: TinyML; Edge Computing; computer vision; passenger counting; ESP32-S3; FOMO; rural transport; IoT.

INTRODUCCIÓN

En los últimos años, el campo de la Inteligencia Artificial ha experimentado una transformación radical con el surgimiento del TinyML, una disciplina de vanguardia que permite ejecutar modelos de aprendizaje automático directamente en microcontroladores de baja potencia (Ray, 2022). A diferencia de los modelos tradicionales que dependen de grandes servidores, el TinyML facilita el procesamiento local en dispositivos compactos como el ESP32-S3, optimizando el consumo energético y la velocidad de respuesta. Esta evolución tecnológica se enmarca en el paradigma del Edge Computing (computación en el borde), el cual permite que la toma de decisiones ocurra en el lugar de origen de los datos, eliminando la dependencia de la nube y garantizando la privacidad en aplicaciones de visión artificial.

Sin embargo, a pesar del potencial de estas tecnologías, el transporte compartido en sectores rurales de Ecuador aún opera bajo métodos tradicionales que carecen de optimización técnica. Un ejemplo crítico se presenta en la ruta Bolívar - El Ángel, operada por la compañía TAXIBOCARCHI, donde la falta de información operativa en tiempo real sobre la demanda de pasajeros genera una constante incertidumbre. Actualmente, el servicio depende de que se complete una cuota mínima de pasajeros para iniciar el recorrido; no obstante, al no existir un sistema que contabilice y notifique automáticamente cuándo se ha alcanzado dicho umbral, se producen tiempos de espera indeterminados y una gestión ineficiente de las unidades. Los sistemas de monitoreo convencionales resultan inviables para este contexto debido a su alto costo e infraestructura compleja. En consecuencia, existe una necesidad urgente de implementar soluciones basadas en hardware restringido que transformen la presencia física de los usuarios en datos operativos útiles, mejorando la eficiencia del servicio y la toma de decisiones de los conductores en zonas desatendidas.

Por lo tanto, este trabajo tuvo como propósito desarrollar un prototipo tecnológico que integre TinyML y Edge Computing para el conteo automático de pasajeros. Con esta iniciativa

se pretendía validar el uso de algoritmos optimizados en determinación de la demanda en tiempo real, brindando una herramienta de coordinación fiable para TAXIBOCARCHI. Con esta investigación se busca dar respuesta a la interrogante sobre cómo mejorar la movilidad rural mediante el despliegue de inteligencia artificial en el borde.

Bajo estas consideraciones, se establecieron los siguientes objetivos para el desarrollo del estudio.

Objetivo General:

- Desarrollar un prototipo de sistema IoT basado en Edge Computing y TinyML mediante el uso del microcontrolador ESP32-S3 para el conteo automático de pasajeros en la parada de la compañía TAXIBOCARCHI, con el fin de mejorar la coordinación y la eficiencia operativa del servicio de transporte rural.

Objetivos Específicos:

- Analizar las tecnologías de visión artificial ligera y las plataformas de hardware de bajo consumo, con la finalidad de seleccionar la configuración más eficiente para operar en entornos rurales con recursos limitados.
- Diseñar la arquitectura del sistema IoT integrando el módulo de captura de imagen en el microcontrolador XIAO ESP32S3 Sense, definiendo el procesamiento local y el protocolo de comunicación para el envío de alertas.
- Implementar el modelo de detección de objetos basado en la arquitectura FOMO (Faster Objects, More Objects), optimizado para identificar y contar personas utilizando dispositivos de capacidad de procesamiento restringida.
- Validar el prototipo del sistema mediante pruebas de funcionamiento para verificar el cumplimiento de métricas de rendimiento, buscando una precisión superior al 85% y una velocidad de respuesta en tiempo real.

El presente documento se estructura en tres (3) capítulos: El primero aborda la consulta bibliográfica y conceptual sobre TinyML, IoT y Edge Computing que fundamenta la investigación. El segundo capítulo se centra en la descripción de las herramientas tecnológicas, la metodología de prototipos y el desarrollo técnico del sistema en la plataforma Edge Impulse. Finalmente, el tercer capítulo presenta los resultados obtenidos tras la validación del prototipo, el análisis de las métricas de rendimiento y el impacto de la solución en la gestión de TAXIBOCARCHI.

CAPÍTULO I

ESTADO DEL ARTE

El presente capítulo constituye el soporte teórico y referencial que fundamenta el desarrollo del prototipo. En primera instancia, se examina el contexto operativo de la compañía TAXIBOCARCHI, destacando su relevancia en la conectividad del sector rural y las particularidades de su modelo de servicio. Posteriormente, la investigación profundiza en los pilares tecnológicos de la propuesta, explorando conceptos clave como los Sistemas Inteligentes de Transporte (ITS) y la arquitectura de Internet de las Cosas (IoT).

Asimismo, se analiza el cambio de paradigma hacia el *Edge Computing* y la implementación de inteligencia artificial integrada mediante *TinyML*, subrayando las ventajas de procesar datos en dispositivos de recursos limitados como el ESP32-S3. Finalmente, se establece un marco de antecedentes mediante la revisión de estudios previos y soluciones similares de visión artificial, lo cual permitió contextualizar la viabilidad y el carácter innovador del sistema de conteo de pasajeros planteado en este trabajo.

1.1. ANTECEDENTES

El desarrollo de soluciones tecnológicas orientadas a la movilidad inteligente ha experimentado una evolución significativa en los últimos años, expandiendo su alcance desde los grandes centros urbanos hacia modelos de transporte rural compartido (*Ridesharing*). De acuerdo con Mugion et al. (2018), la integración de Sistemas Inteligentes de Transporte (ITS) y Sistemas Avanzados de Información para Viajeros (ATIS) es fundamental para optimizar la logística operativa, mejorar la toma de decisiones y reducir las brechas de conectividad en zonas geográficamente dispersas. Sin embargo, la implementación de estas tecnologías en sectores rurales enfrenta desafíos críticos debido a las limitaciones de infraestructura y la falta de conectividad constante a internet.

Para superar estas barreras operativas, la literatura científica reciente evidencia una clara transición metodológica: el abandono progresivo de las arquitecturas centralizadas en la nube (*Cloud Computing*) en favor del procesamiento en el borde (*Edge Computing*). Según Shafique et al. (2020), esta tendencia responde a la exigencia imperativa de reducir la latencia, minimizar el consumo de ancho de banda y garantizar la privacidad de los usuarios mediante el procesamiento local de la información. En concordancia con esta postura, investigaciones de autores como Merenda et al. (2020) y Ray (2022) sostienen que la convergencia del Internet de las Cosas (IoT) con el aprendizaje automático integrado (*TinyML*) representa la solución técnica óptima para entornos restringidos. Estos autores coinciden en que trasladar la capacidad de inferencia algorítmica directamente a microcontroladores de ultra bajo consumo permite ejecutar tareas complejas de visión artificial en el mismo lugar donde se generan los datos, mitigando la dependencia de servidores externos.

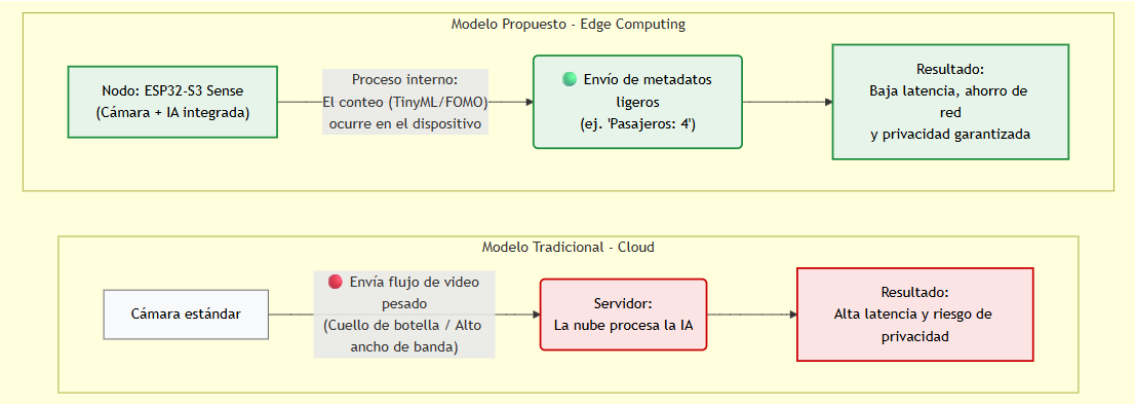
Bajo esta premisa, la revisión de los trabajos previos expuestos a continuación permite contextualizar la viabilidad técnica y operativa de aplicar estos paradigmas tecnológicos en la gestión de flotas vehiculares. El análisis de estas investigaciones establece una base sólida, comparativa y debidamente fundamentada para el diseño del sistema automatizado propuesto en la compañía TAXIBOCARCHI.

1.1.1. Arquitecturas de *Edge Computing* para el conteo de pasajeros y gestión del transporte

En el ámbito de la optimización del transporte mediante sistemas de visión artificial, investigaciones recientes han demostrado la eficacia operativa de descentralizar el procesamiento de imágenes. Un estudio destacado, publicado por Laguna et al. (2025) en la revista *Sensors*, analizó el diseño e implementación de un sistema embebido para el conteo de pasajeros en estaciones de transporte masivo. Los investigadores determinaron que el envío continuo de flujos de video hacia servidores centralizados generaba cuellos de botella críticos

y latencias inaceptables para la toma de decisiones en tiempo real. Para resolver esta problemática, propusieron una arquitectura descentralizada donde la captura de imágenes y la estimación numérica de los usuarios se realizaron de forma local en el nodo sensor, enviando a la red únicamente metadatos estructurados (fecha, hora, ubicación y cantidad de pasajeros). Esta investigación valida directamente la viabilidad de la arquitectura propuesta para la compañía TAXIBOCARCHI, demostrando que la transmisión exclusiva de datos ligeros es la estrategia idónea para gestionar la demanda dinámica sin saturar las limitadas infraestructuras de comunicación del entorno rural, tal como se observa en la Figura 1.

Figura 1. *Comparativa de procesamiento Cloud vs. Edge Computing*



Nota: Adaptado de Laguna et al. (2025).

Este enfoque de procesamiento en sitio es fuertemente respaldado por investigaciones como la de Nfissi et al. (2024), presentada en la *IEEE International Conference on Semantic Computing*. En dicho foro científico, se evaluó una solución de *Edge Computing* orientada al conteo de pasajeros en paradas de autobuses con restricciones energéticas. El estudio concluyó que el despliegue de modelos de detección de objetos directamente en el dispositivo de borde no solo redujo el consumo de memoria y ancho de banda en más de un 50% en comparación con los métodos tradicionales, sino que resultó ser una solución éticamente responsable. Al procesar las imágenes localmente y descartarlas de manera inmediata tras la extracción de los datos numéricos, el sistema garantizó la privacidad absoluta de los usuarios. Este antecedente resultó de vital importancia para el presente trabajo, ya que permitió justificar tanto técnica

como éticamente la decisión de no transmitir ni almacenar registros fotográficos de los habitantes de la ruta Bolívar - El Ángel, asegurando así la viabilidad social y la adopción del prototipo.

1.1.2. Optimización de algoritmos de visión artificial mediante *TinyML* y arquitectura FOMO

La implementación de visión artificial en dispositivos de borde requiere una reingeniería profunda de los modelos de aprendizaje profundo tradicionales, los cuales han sido históricamente diseñados para entornos con alta capacidad de cómputo. En este contexto, investigaciones recientes destacan que la clave para el éxito del *TinyML* reside en la optimización algorítmica orientada a restricciones. Arsalan et al. (2024) desarrollaron un estudio sobre la detección de personas para aplicaciones de edificios inteligentes, donde demostraron que, mediante el uso de técnicas combinadas de cuantización (*quantization*) y poda de redes (*pruning*), es posible desplegar modelos de detección en microcontroladores con recursos extremadamente limitados sin sacrificar significativamente la precisión. Los autores concluyeron que el procesamiento local no solo mejora la eficiencia energética, sino que permite una respuesta en tiempo real que sería inalcanzable mediante el envío de datos a la nube en entornos con alta densidad de usuarios.

Entre estas técnicas de optimización, la cuantización (quantization) es una de las de mayor impacto para el despliegue en microcontroladores. Consiste en reducir la precisión numérica con la que se representan los pesos y las activaciones del modelo: en lugar de emplear números de punto flotante de 32 bits (float32), propios de la fase de entrenamiento, los valores se convierten a enteros de 8 bits (int8). Esta conversión reduce aproximadamente cuatro veces el tamaño del modelo en memoria y acelera los cálculos, ya que la aritmética de enteros resulta más eficiente en procesadores que carecen de una unidad de punto flotante dedicada, como el ESP32-S3. A cambio de una pérdida de precisión por lo general mínima, la cuantización int8

es la que permite que un modelo de visión artificial quepa y se ejecute dentro de las decenas de kilobytes de memoria RAM disponibles en este tipo de hardware (Arsalan et al., 2024).

Complementariamente, la adaptación de las arquitecturas de redes neuronales en función de las limitaciones físicas del hardware es un área de estudio crítica para la ingeniería actual. Sudhakar et al. (2023) investigaron la adaptación de arquitecturas impulsada por restricciones para la localización de imágenes en robots miniatura, un escenario técnico análogo al uso del microcontrolador ESP32-S3. Su investigación resalta que la selección de una arquitectura debe estar supeditada a un equilibrio estricto entre la latencia de inferencia y la disponibilidad de memoria RAM. Este enfoque de "diseño consciente del hardware" respalda la elección de arquitecturas no convencionales para tareas de conteo, donde la prioridad es la velocidad de ejecución en el borde.

Bajo estos principios de optimización y adaptación técnica, la arquitectura FOMO (*Faster Objects, More Objects*) se presenta como la solución idónea para el conteo de pasajeros en entornos de hardware restringido. A diferencia de los detectores de objetos tradicionales que generan cajas delimitadoras complejas y consumen amplios recursos de memoria, la literatura técnica establece que FOMO simplifica el problema de visión transformándolo en una tarea de localización de centroides basada en celdas de cuadrícula. Este cambio de paradigma permite que el sistema detecte y contabilice múltiples objetivos en tiempo real con una fracción del uso de memoria de modelos como YOLO, alineándose con las metodologías de optimización propuestas por Arsalan et al. (2024) y garantizando la viabilidad operativa del prototipo en la parada de TAXIBOCARCHI.

1.1.3. Capacidades de inferencia en hardware de ultra bajo consumo (Familia ESP32)

Históricamente, los microcontroladores (MCU) fueron diseñados exclusivamente para tareas de lógica de control básica y lectura de sensores, careciendo de la capacidad de procesamiento necesaria para ejecutar modelos de aprendizaje automático. Sin embargo, la

evolución de los sistemas embebidos ha permitido la integración de unidades de procesamiento digital de señales (DSP) e instrucciones vectoriales en chips de ultra bajo consumo, transformando dispositivos como el ESP32-S3 en plataformas viables para la inferencia de Inteligencia Artificial en el borde.

La capacidad de la arquitectura ESP32 para gestionar tareas complejas de visión artificial bajo restricciones energéticas ha sido sólidamente documentada en la literatura reciente. Un estudio de alta relevancia desarrollado por Novak et al. (2024), publicado en la revista *Engineering Science and Technology*, validó el uso de esta familia de microcontroladores para la detección de objetos en tiempo real en entornos naturales aislados. En su investigación, los autores diseñaron una sonda de inspección autónoma basada en IoT para el monitoreo forestal, logrando adaptar e incrustar con éxito el algoritmo FOMO dentro del microcontrolador ESP32. El estudio demostró empíricamente que este hardware posee la eficiencia computacional requerida para procesar imágenes de forma rápida y continua, operando de manera estable con un consumo energético mínimo.

Estos hallazgos científicos fueron directamente extrapolables a los requerimientos operativos de la parada de transporte en la ruta Bolívar - El Ángel. La evidencia proporcionada por Novak et al. (2024) fundamenta y justifica la selección del microcontrolador XIAO ESP32S3 Sense para el presente proyecto. Al comprobarse que la familia ESP32 es capaz de sostener modelos de detección espacial (FOMO) en despliegues de campo (*embedded IoT*), se garantiza que el dispositivo seleccionado para la compañía TAXIBOCARCHI tendrá la robustez y capacidad de memoria necesarias para realizar el conteo ininterrumpido de pasajeros, superando las limitaciones de los sistemas tradicionales.

1.1.4. Protocolos de comunicación y transmisión de datos en entornos restringidos

La efectividad de un sistema basado en *Edge Computing* no depende de manera exclusiva de su capacidad de inferencia algorítmica local, sino también de la eficiencia con la

que logra transmitir los resultados procesados hacia los usuarios finales. En zonas geográficamente dispersas o rurales, donde la intermitencia y el bajo ancho de banda de las redes de comunicación representan un desafío crítico, el diseño de la arquitectura de red debe priorizar el intercambio de información ultraligera.

En este contexto, la literatura científica sobre el Internet de las Cosas (IoT) destaca la adopción de arquitecturas orientadas a eventos. Investigaciones enfocadas en la telemetría y el monitoreo remoto, como la desarrollada por Mokhtar et al. (2023), han evaluado el rendimiento de transmitir datos estructurados en formato JSON (*JavaScript Object Notation*) a través de protocolos web estándar para sistemas de alerta temprana. El estudio demostró que encapsular la información en paquetes de texto plano y utilizar *Webhooks* o Interfaces de Programación de Aplicaciones (API) de plataformas de mensajería instantánea (como Telegram) reduce el consumo de datos a unos pocos kilobytes por transacción. Los autores concluyeron que este método garantiza la entrega de notificaciones *push* de baja latencia directamente a los dispositivos móviles de los usuarios, evadiendo la necesidad de desarrollar y mantener servidores dedicados o aplicaciones de terceros que consuman recursos adicionales.

El uso de una comunicación asíncrona y de bajo peso de datos resulta idóneo dadas las características de la infraestructura de red en la ruta Bolívar - El Ángel. La evidencia proporcionada por estos estudios justifica el diseño de red propuesto para el prototipo, en el cual el ESP32 no transmite telemetría constante, sino que actúa por interrupciones; es decir, emite una alerta estructurada a través de la API únicamente en el instante en que el algoritmo FOMO verifique que se cumple la cuota mínima de pasajeros en la parada. Esta estrategia de comunicación asegura que los conductores de la compañía TAXIBOCARCHI reciban datos logísticos oportunos y fiables para iniciar su recorrido, mitigando los problemas de conectividad del entorno rural y cerrando el ciclo operativo del sistema inteligente.

1.2. MARCO CONTEXTUAL INSTITUCIONAL Y OPERATIVO: LA COMPAÑÍA TAXIBOCARCHI

1.2.1. Caracterización Operativa, Modelo de Gestión y Brechas Logísticas

La Compañía de Transporte TAXIBOCARCHI es una operadora comercial legalmente constituida que ejerce sus actividades en la provincia del Carchi bajo la categoría de transporte comercial rural. Su funcionamiento se rige por la Ley Orgánica de Transporte Terrestre, Tránsito y Seguridad Vial (LOTTTSV), específicamente bajo la Ley Orgánica Reformatoria de 2021, la cual delega a estas operadoras la responsabilidad de asegurar la conectividad en parroquias y recintos con cobertura limitada de transporte público masivo (Asamblea Nacional del Ecuador, 2021). En la región andina, este rol trasciende lo comercial; la compañía actúa como un enlace estratégico para la cadena de suministro local y el acceso a derechos fundamentales como la salud y la educación, lo que obliga a la institución a someterse a controles de la Agencia Nacional de Tránsito (ANT) para garantizar eficiencia y seguridad en sus rutas (Instituto Nacional de Estadística y Censos [INEC], 2022).

Para cumplir con este mandato social y asegurar la sostenibilidad financiera en zonas de baja densidad poblacional, la organización implementa en el eje Bolívar - El Ángel un modelo logístico conocido como Transporte Sensible a la Demanda o DRT (*Demand-Responsive Transport*). A diferencia de los sistemas urbanos con horarios rígidos, el modelo DRT es la respuesta técnica a la baja afluencia de usuarios: las unidades de "taxi-ruta" no inician su recorrido en intervalos fijos, sino que se despachan únicamente cuando se ha reunido una cuota mínima de pasajeros que cubra los costos de operación (Vasconcellos, 2020). Este esquema colaborativo exige que los pasajeros se congreguen en puntos de convergencia o paradas y esperen el aforo máximo del vehículo, lo que minimiza el costo por kilómetro para los conductores. No obstante, la eficiencia de esta gestión depende de una variable crítica: el

flujo de información constante y preciso entre la demanda en la parada y la oferta representada por el conductor (Navarro et al., 2021).

Actualmente, TAXIBOCARCHI enfrenta un déficit significativo de comunicación en esta dinámica, operando bajo lo que se denomina una "zona de sombra de datos". Al no contar con un sistema de telemetría o monitoreo en tiempo real, la recolección de pasajeros se realiza de manera empírica, lo que incrementa el "kilometraje muerto" (recorridos sin pasajeros), desperdicia combustible y eleva la huella de carbono de la flota (Cruz & Katz, 2022). Asimismo, esta incapacidad de anticipar la afluencia de personas genera tiempos de espera prolongados para los usuarios, deteriorando la calidad del servicio. Ante estas brechas logísticas, la digitalización de la infraestructura mediante sensores de bajo costo se vuelve imperativa para convertir esta red en un Sistema Inteligente de Transporte (ITS). Según García-Albertos et al. (2023), la implementación de microcontroladores y algoritmos de visión artificial (*TinyML*) permite censar la demanda en tiempo real, eliminando la incertidumbre y permitiendo que los conductores despachen las unidades basándose en datos numéricos exactos.

1.3. MOVILIDAD INTELIGENTE Y SISTEMAS DE TRANSPORTE

1.3.1. Movilidad rural y el fenómeno del desajuste espacial

La movilidad en entornos rurales se define por la interacción entre la infraestructura vial limitada y la dispersión de los puntos de origen y destino. A diferencia de la planificación urbana, la movilidad rural enfrenta el fenómeno técnico del "desajuste espacial" (*spatial mismatch*), el cual ocurre cuando las oportunidades de servicios y empleo se encuentran geográficamente aisladas de las zonas residenciales de baja densidad. De acuerdo con Porru et al. (2020), esta desconexión genera una dependencia crítica de sistemas de transporte no convencionales, ya que las redes de tránsito masivo resultan económicamente inviables debido a los bajos coeficientes de ocupación por kilómetro recorrido. En este escenario, la ingeniería

de transporte busca modelos que logren mitigar esta brecha mediante la optimización de los recursos móviles existentes.

1.3.2. Modelos de transporte compartido

El transporte compartido, o *Ridesharing*, constituye una estrategia de movilidad colaborativa diseñada para maximizar la eficiencia de los vehículos de baja capacidad. Bajo este modelo, el sistema agrupa a múltiples usuarios con trayectorias coincidentes en una sola unidad, reduciendo el costo operativo individual y el impacto ambiental del desplazamiento. Según Vasconcellos (2020), en el contexto de las periferias andinas, el transporte compartido deja de ser una opción de conveniencia para transformarse en una infraestructura de movilidad esencial. Para que este modelo sea sostenible, requiere una gestión de aforo precisa que evite la subutilización del vehículo, fundamentando la necesidad de sistemas de conteo automatizados que informen sobre la ocupación real en cada nodo de la ruta.

1.3.3. Sistemas Inteligentes de Transporte (ITS)

Los Sistemas Inteligentes de Transporte (ITS, por sus siglas en inglés: *Intelligent Transportation Systems*) representan la aplicación de tecnologías de la información y comunicación (TIC) para mejorar la seguridad, eficiencia y sostenibilidad del transporte. Un ITS no se limita a la gestión del tráfico, sino que engloba una arquitectura de red donde los vehículos, la infraestructura y los usuarios intercambian datos en tiempo real. Para García-Albertos et al. (2023), la transición hacia un ITS en sectores rurales permite convertir paradas pasivas en nodos inteligentes de recolección de datos, facilitando una administración de flota basada en la demanda real y no en estimaciones empíricas. La implementación de visión artificial en estos sistemas es el avance más reciente, permitiendo el procesamiento de datos visuales directamente en el "borde" de la infraestructura.

1.3.4. Sistemas Avanzados de Información para Viajeros (ATIS)

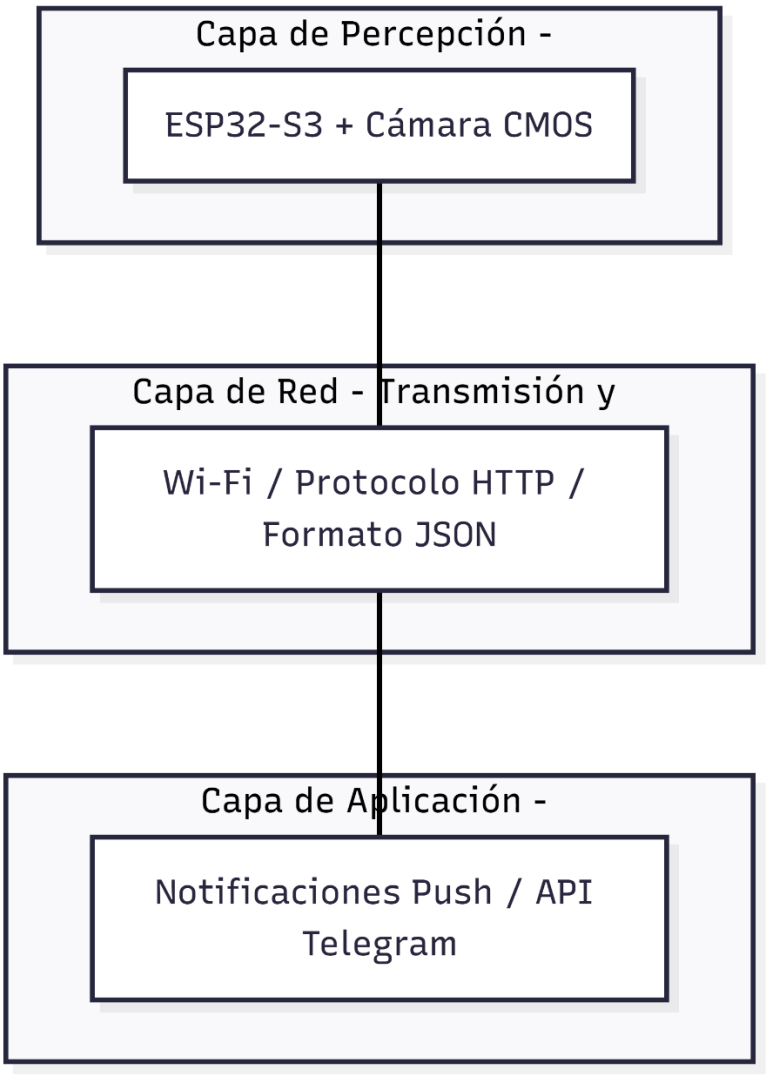
Dentro del ecosistema ITS, los Sistemas Avanzados de Información para Viajeros (ATIS, por sus siglas en inglés: *Advanced Traveler Information Systems*) actúan como la capa de salida y comunicación del sistema. Su función es procesar la información recolectada por los sensores y entregarla de forma procesada a los conductores y pasajeros para mejorar la toma de decisiones. Un ATIS eficiente reduce la incertidumbre logística al proporcionar alertas tempranas sobre la disponibilidad de espacios o la presencia de pasajeros en puntos remotos. Investigaciones recientes sostienen que la eficacia de un ATIS en entornos de baja conectividad depende de protocolos de mensajería ligeros que aseguren la entrega de la información con la mínima latencia posible (Samuda et al., 2023).

1.4. INFRAESTRUCTURA DE REDES Y PARADIGMAS DE COMPUTACIÓN

1.4.1. Arquitectura de Internet de las Cosas (IoT)

El Internet de las Cosas (IoT, por sus siglas en inglés: *Internet of Things*) es un paradigma tecnológico que expande la conectividad de la red más allá de los ordenadores tradicionales, integrando objetos físicos equipados con sensores y capacidad de procesamiento. Para garantizar la interoperabilidad y el flujo eficiente de datos, la literatura de ingeniería define el ecosistema IoT mediante un modelo clásico de tres capas funcionales, tal como se ilustra en la Figura 2. En primer lugar, la capa de percepción es la encargada de la adquisición de señales del entorno físico a través de sensores y microcontroladores. En segundo lugar, la capa de red asume la responsabilidad de la transmisión de la telemetría mediante protocolos inalámbricos y la estructuración de paquetes de datos. Finalmente, la capa de aplicación consolida la interfaz lógica donde los datos procesados se transforman en servicios o alertas utilizables para el usuario final (Shafique et al., 2020).

Figura 2. *Arquitectura de tres niveles del Internet de las Cosas (IoT)*



Nota: El diagrama representa la jerarquía funcional del sistema, desde la captura de datos en el borde hasta la entrega de información al conductor. Adaptado de Shafique et al. (2020).

1.4.2. Transición del *Cloud Computing* al *Edge Computing*

Históricamente, la gestión de datos en sistemas IoT se ha fundamentado en el *Cloud Computing* (computación en la nube), donde los dispositivos actúan como terminales "ciegas" que envían información cruda (como flujos de video) a servidores centralizados para su procesamiento. Sin embargo, en entornos rurales con conectividad intermitente, este modelo resulta ineficiente debido a su alta latencia, consumo excesivo de ancho de banda y vulnerabilidades en la privacidad de los usuarios (Merenda et al., 2020).

Para solventar estas limitaciones, la ingeniería actual ha pasado hacia el paradigma del *Edge Computing* (computación en el borde).

1.4.3. Protocolos de comunicación y estructuración de datos

Al procesar la información en el borde de la red, el sistema solo requiere transmitir los resultados estructurados hacia la capa de aplicación. En ecosistemas con restricciones energéticas y de red, se descartan los lenguajes de marcado pesados y redundantes (como XML) a favor de formatos ultraligeros, estableciéndose el uso de JSON (*JavaScript Object Notation*) como el estándar de la industria para este propósito.

Este protocolo consiste en un formato de texto plano basado en pares de clave-valor que prescinde de las complejas etiquetas de cierre de su predecesor, lo cual permite que el hardware encapsule la información vital de manera directa (por ejemplo: {"parada": "El Ángel", "pasajeros": 4}). Esta abstracción permite transmitir la alerta consumiendo apenas unos pocos *bytes* de ancho de banda, factor que resulta crítico para mantener la estabilidad de la conexión en zonas de cobertura celular intermitente. En este sentido, investigaciones comparativas sobre el rendimiento de protocolos en ecosistemas IoT demuestran que JSON supera significativamente a XML al reducir el tamaño de la carga útil (*payload*) de los paquetes de datos entre un 30% y un 50%, a la vez que incrementa la velocidad de decodificación en los microcontroladores (Shafique et al., 2020).

1.4.4. Mensajería automatizada mediante API y *Webhooks*

El último eslabón de la infraestructura de red es la entrega de la información al usuario final (el conductor). Para evitar los costos operativos, la latencia y la complejidad asociados al desarrollo y mantenimiento de servidores web dedicados o aplicaciones móviles propietarias, la arquitectura del sistema aprovecha el uso de Interfaces de Programación de Aplicaciones (API) de plataformas de mensajería instantánea de terceros.

Mediante el uso de *Webhooks* (solicitudes HTTP/REST asíncronas), el microcontrolador puede comunicarse directamente con los servidores de estas plataformas. De acuerdo con estudios experimentales de telemetría, como el desarrollado por Samuda et al. (2023), la implementación de *bots* de Telegram gestionados mediante API en prototipos IoT de bajo costo ha demostrado ser una solución de grado industrial altamente eficiente. El estudio concluye que este modelo no solo garantiza la entrega bidireccional y encriptada de notificaciones *push* con retardos inferiores a un segundo, sino que evade por completo la necesidad de aprovisionar infraestructura en la nube. En el contexto de TAXIBOCARCHI, esta estrategia garantiza que la alerta llegue directamente al teléfono inteligente del transportista en el instante exacto en que se cumple la cuota de pasajeros, logrando la máxima eficiencia logística con el mínimo costo computacional.

1.5. INTELIGENCIA ARTIFICIAL Y APRENDIZAJE AUTOMÁTICO INTEGRADO

1.5.1. Deep Learning y Visión Artificial: Conceptos base

La Inteligencia Artificial (IA) ha evolucionado desde sistemas basados en reglas hacia el Aprendizaje Automático (*Machine Learning*), donde los algoritmos identifican patrones en los datos de forma autónoma. Dentro de este campo, el Aprendizaje Profundo (*Deep Learning* - DL) utiliza arquitecturas inspiradas en la estructura biológica del cerebro para procesar información no estructurada. La Visión Artificial se beneficia directamente del DL al permitir que una máquina "vea" e interprete imágenes mediante la descomposición de estas en matrices numéricas de píxeles, identificando jerarquías de características que van desde bordes simples hasta objetos complejos. En el contexto de la gestión de transporte, esta capacidad permite la transición de cámaras de vigilancia pasivas a sensores inteligentes capaces de realizar inferencias sobre la ocupación vehicular en tiempo real (Laguna et al., 2025).

1.5.2. Tiny Machine Learning (TinyML): Optimización para el borde

Históricamente, los modelos de DL requerían hardware con alta capacidad de cómputo y memoria. Sin embargo, el surgimiento del *Tiny Machine Learning* (*TinyML*) ha permitido trasladar estas capacidades a microcontroladores de ultra bajo consumo. Esta tecnología se fundamenta en la optimización algorítmica mediante dos técnicas críticas: la cuantización y la poda. La cuantización reduce la precisión de los pesos de la red neuronal (por ejemplo, de punto flotante de 32 bits a enteros de 8 bits), disminuyendo drásticamente el uso de memoria sin una pérdida significativa de precisión. Por su parte, la poda elimina las conexiones neuronales que no contribuyen al resultado final, aligerando el modelo. Según Ray (2022), el *TinyML* es el habilitador clave para sistemas de detección de personas en edificios inteligentes y paradas de transporte, permitiendo que la inferencia ocurra en el dispositivo sin depender de servidores externos.

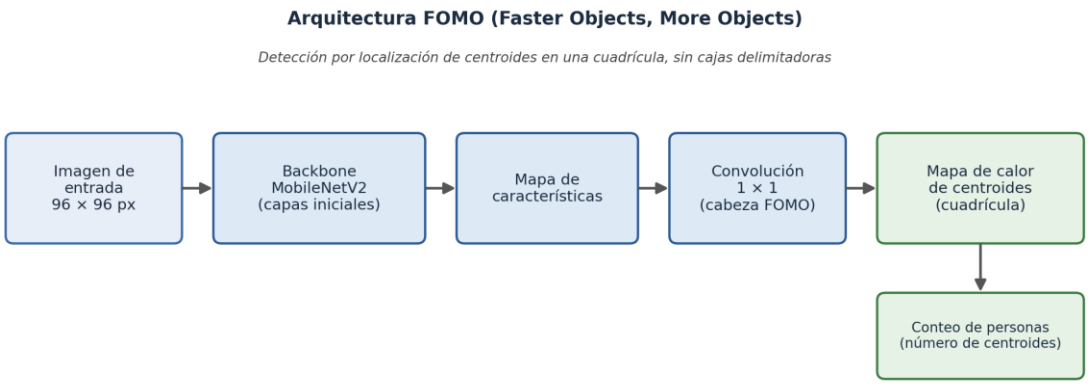
1.5.3. Arquitectura de detección FOMO (Faster Objects, More Objects)

Para aplicaciones de conteo en hardware restringido como el ESP32-S3, la arquitectura FOMO se presenta como la solución técnica más eficiente. A diferencia de detectores convencionales como YOLO (*You Only Look Once*), que intentan predecir cajas delimitadoras complejas que consumen recursos excesivos de memoria RAM, FOMO simplifica la detección de objetos transformándola en una tarea de localización de centroides basada en una cuadrícula, como se ilustra en la Figura 3.

El algoritmo divide la imagen de entrada en celdas y predice la probabilidad de que un objeto esté presente en el centro de cada una de ellas. Esta metodología permite que el sistema detecte múltiples objetivos simultáneamente con una fracción del costo computacional de los modelos tradicionales. Investigaciones recientes han demostrado que FOMO optimizado para computación en el borde en microcontroladores puede alcanzar una alta velocidad de procesamiento (FPS) manteniendo una precisión adecuada para tareas de inspección y conteo

(Lin et al., 2024). Para la compañía TAXIBOCARCHI, esto significa que el prototipo puede identificar y cuantificar pasajeros de forma continua, operando con la memoria limitada del dispositivo y asegurando una respuesta inmediata.

Figura 3. *Esquema de la arquitectura de detección FOMO*



Nota. El modelo procesa la imagen de entrada a través del backbone MobileNetV2 y, mediante una convolución 1 × 1, genera un mapa de calor de centroides sobre una cuadrícula. El conteo de personas se obtiene a partir del número de centroides detectados, sin generar cajas delimitadoras, lo que reduce el costo computacional y el uso de memoria.

1.6. SISTEMAS EMBEBIDOS Y HARDWARE DE PROCESAMIENTO

1.6.1. Microcontroladores y sistemas *System on a Chip* (SoC)

La implementación física de arquitecturas de computación en el borde en entornos rurales requiere plataformas de hardware que equilibren capacidad de cómputo, bajo consumo energético y dimensiones reducidas. Este requerimiento se satisface mediante el uso de sistemas embebidos, y específicamente, mediante la tecnología *System on a Chip* (SoC). Un SoC es un circuito integrado que encapsula todos los componentes electrónicos necesarios para un sistema informático (unidad central de procesamiento, memoria, puertos de entrada/salida y módulos de radiofrecuencia) en un único sustrato de silicio. De acuerdo con Merenda et al. (2020), la adopción de arquitecturas SoC en nodos IoT elimina la latencia de comunicación

entre chips independientes y reduce drásticamente el consumo de potencia, lo cual resulta un factor crítico para el despliegue de sensores autónomos en las paradas de la cooperativa.

1.6.2. Arquitectura del microcontrolador ESP32-S3

El núcleo de procesamiento seleccionado para el prototipo es el microcontrolador ESP32-S3, un SoC diseñado específicamente para cargas de trabajo de Inteligencia Artificial en el borde del Internet de las Cosas. A diferencia de los microcontroladores de propósito general, el ESP32-S3 integra un procesador de doble núcleo Xtensa de 32 bits (LX7) que incorpora extensiones de instrucciones vectoriales orientadas a acelerar redes neuronales y un módulo de Procesamiento Digital de Señales (DSP).

Esta arquitectura de hardware es la que hace matemáticamente posible la ejecución del algoritmo FOMO. Las instrucciones vectoriales permiten al chip realizar múltiples operaciones matemáticas simultáneas (Single Instruction, Multiple Data - SIMD), acelerando drásticamente los cálculos de las matrices convolucionales generadas por el modelo de *TinyML* (Lin et al., 2024). Sin esta aceleración nativa por hardware, el procesamiento de imágenes causaría cuellos de botella térmicos y computacionales, impidiendo lograr los *Frames Per Second* (FPS) necesarios para realizar el conteo de pasajeros en tiempo real.

1.6.3. Sistemas de percepción visual

Para alimentar las redes neuronales ejecutadas por el SoC, el sistema requiere un mecanismo de adquisición de datos del entorno físico. Esta tarea recae en los sensores ópticos de tecnología CMOS (*Complementary Metal-Oxide-Semiconductor*). A diferencia de los sistemas de captura fotográfica convencionales, un sensor CMOS para visión artificial opera mediante una matriz de fotodiodos que convierten la intensidad lumínica entrante directamente en señales de voltaje en cada píxel de forma independiente y paralela.

El uso de tecnología CMOS está justificado en la ingeniería de prototipos de internet de las cosas debido a su alta velocidad de lectura, fácil integración con los puertos paralelos

del microcontrolador y bajo consumo eléctrico en comparación con arquitecturas predecesoras (como los sensores CCD). Según Laguna et al. (2025), la combinación de sensores CMOS de bajo coste con procesamiento neuronal en el borde constituye la base del monitoreo eficiente en infraestructuras de transporte, ya que permite digitalizar el escenario de la parada de autobuses en matrices numéricas ligeras que el ESP32-S3 puede procesar e interpretar de forma inmediata.

1.7. INGENIERÍA DE SOFTWARE Y MÉTRICAS DE EVALUACIÓN

1.7.1. Metodología de desarrollo (Modelo de Prototipos)

La construcción de sistemas embebidos orientados al Internet de las Cosas requiere un enfoque metodológico que permita la integración continua entre componentes físicos y lógicos. Para este proyecto, se adopta el Modelo de Desarrollo por Prototipos. De acuerdo con Pressman y Maxim (2020), a diferencia de las metodologías secuenciales clásicas, este modelo iterativo se fundamenta en la construcción de versiones preliminares y ejecutables del sistema (prototipos) para comprender y refinar los requisitos tecnológicos antes del desarrollo a gran escala.

Esta aproximación iterativa ha demostrado ser altamente efectiva en arquitecturas IoT modernas. Según investigaciones recientes sobre metodologías de desarrollo en ecosistemas de conectividad rural, el uso de plataformas de prototipado permite a los ingenieros evaluar la viabilidad de la recolección de datos y el procesamiento local (*Edge Computing*) en condiciones adversas (Guevara et al., 2024). En el contexto de la empresa TAXIBOCARCHI, el despliegue mediante prototipos es fundamental; permite someter al hardware a pruebas en entornos controlados para identificar tempranamente limitaciones de memoria en el microcontrolador ESP32-S3, cuellos de botella térmicos y fallos de calibración en los sensores ópticos, asegurando la confiabilidad del sistema antes de su implementación final en las paradas de la ruta Bolívar - El Ángel.

1.7.2. Metodología ágil Scrum

El desarrollo de aplicaciones de software modernas, y en particular los sistemas web orientados a la visualización de datos en tiempo real, requiere un enfoque metodológico que priorice la entrega incremental de valor y la adaptación continua ante cambios en los requisitos. En este contexto, Scrum se ha consolidado como el marco de trabajo ágil más ampliamente adoptado en la industria del software. De acuerdo con Schwaber y Sutherland (2020), Scrum es un marco ligero que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos, estableciendo que la suma de todos los eventos Scrum forma el Sprint, el cual actúa como contenedor de los demás eventos.

La estructura de Scrum se articula en torno a tres pilares fundamentales: la transparencia, la inspección y la adaptación. Estos principios garantizan que todos los aspectos del proceso de desarrollo sean visibles para los responsables del resultado, que los artefactos y el progreso sean evaluados con frecuencia, y que el equipo ajuste su enfoque cuando detecta desviaciones respecto a los objetivos definidos (Schwaber y Sutherland, 2020).

Roles del equipo Scrum

El equipo Scrum se compone de tres roles bien definidos. El Product Owner es el responsable de maximizar el valor del producto, gestionando y priorizando el Product Backlog. El Scrum Master facilita el proceso, eliminando impedimentos y asegurando que el equipo comprenda y aplique correctamente el marco de trabajo. Finalmente, los Developers son quienes diseñan, construyen y verifican los incrementos del producto durante cada Sprint.

Artefactos de Scrum

Los tres artefactos principales de Scrum son el Product Backlog, el Sprint Backlog y el Incremento. El Product Backlog es una lista ordenada y dinámica de todo lo que se conoce que es necesario para el producto, gestionada exclusivamente por el Product Owner y compuesta por historias de usuario priorizadas. El Sprint Backlog contiene los elementos del Product

Backlog seleccionados para el Sprint en curso, junto con el plan para entregar el Incremento. El Incremento es el resultado concreto y verificable de cada Sprint, el cual debe cumplir la Definición de Terminado para ser considerado entregable (Schwaber y Sutherland, 2020).

Eventos de Scrum

Scrum define cinco eventos formales que estructuran el trabajo iterativo. El Sprint es el contenedor principal con duración fija de dos semanas como máximo, dentro del cual se ejecutan los demás eventos. El Sprint Planning define qué se construirá durante el Sprint y cómo se logrará. El Daily Scrum es una sincronización diaria de 15 minutos para inspeccionar el progreso. El Sprint Review evalúa el Incremento con los interesados al finalizar el Sprint. Finalmente, la Sprint Retrospective permite al equipo reflexionar sobre el proceso y proponer mejoras para el Sprint siguiente.

La aplicación de Scrum para el desarrollo del panel web de monitoreo de TAXIBOCARCHI respondió a la naturaleza iterativa del componente de software, cuya funcionalidad debió refinarse en ciclos cortos para alinear la interfaz con las necesidades reales del administrador de flota, en complemento con el modelo de prototipos empleado para el componente de hardware IoT.

1.7.3. Plataformas de desarrollo *Machine Learning as a Service*

El ciclo de vida del *TinyML* se divide en dos fases computacionalmente asimétricas: el entrenamiento del modelo (que requiere alto poder de procesamiento) y la inferencia (que se ejecuta en hardware restringido). Para gestionar este ecosistema, la arquitectura de software se apoya en plataformas *Machine Learning as a Service* (MLaaS), destacando el rol de Edge Impulse. Esta plataforma en la nube permite la ingesta de los datos fotográficos, la extracción de características y el entrenamiento del algoritmo FOMO utilizando servidores de alto rendimiento. Posteriormente, Edge Impulse empaqueta la red neuronal entrenada en una biblioteca de lenguaje C++ optimizada para microcontroladores (Ray, 2022). Finalmente, la

integración del modelo de IA con los protocolos de red Wi-Fi y la estructuración del carga útil JSON se programa e implanta en el firmware del dispositivo a través del Entorno de Desarrollo Integrado (IDE) de Arduino.

1.7.4. Métricas de evaluación algorítmica

Para garantizar que el sistema de TAXIBOCARCHI sea logísticamente viable, la red neuronal debe someterse a una evaluación cuantitativa mediante métricas de rendimiento de clasificación. La métrica principal es la Precisión (*Accuracy*), que representa el porcentaje de predicciones correctas sobre el total de inferencias realizadas. Sin embargo, en un entorno dinámico, es crítico analizar la Matriz de Confusión del algoritmo.

En este contexto, se debe minimizar la tasa de falsos positivos (el sistema detecta a un pasajero cuando en realidad es una sombra o un objeto inanimado), ya que esto provocaría el despacho de unidades vacías, incrementando el kilometraje muerto. Simultáneamente, se debe controlar la tasa de falsos negativos (el sistema ignora a una persona real en la parada), lo cual dejaría a los usuarios sin servicio (Laguna et al., 2025).

1.7.5. Métricas de rendimiento de hardware

Más allá de la exactitud matemática del modelo, un sistema ITS debe operar en una ventana de tiempo que permita la toma de decisiones. Por ello, el desempeño del microcontrolador se evalúa mediante dos métricas físicas. La primera es la Velocidad de Inferencia, medida en *Frames Per Second* (FPS), que indica cuántas imágenes completas puede capturar y procesar el ESP32-S3 en un segundo. La segunda es la Latencia, definida como el tiempo de retardo en milisegundos (ms) desde que el sensor CMOS captura el fotón hasta que la capa de aplicación emite el resultado estructurado. Investigaciones aplicadas como la de Lin et al. (2024) documentan que la combinación del algoritmo FOMO y hardware acelerado por DSP en microcontroladores de gama media logra latencias del orden de cientos de

milisegundos, suficientes para garantizar el monitoreo de la demanda en tiempo real en escenarios de conteo estático.

CAPÍTULO II

MATERIALES Y MÉTODOS

En este capítulo se definen las pautas metodológicas, procedimientos y técnicas aplicadas para alcanzar los objetivos de la investigación. Se expone el diseño del estudio y los métodos de recolección de datos empleados en la ruta Bolívar - El Ángel para abordar el problema logístico de la compañía TAXIBOCARCHI. Por tratarse de un proyecto de innovación tecnológica, se detalla específicamente el modelo de desarrollo utilizado para el diseño, entrenamiento algorítmico, implementación de hardware y validación del prototipo IoT. Esta estructura metodológica garantizó la rigurosidad técnica de cada fase y aseguró que los resultados respondieran de manera efectiva a las necesidades operativas de la operadora de transporte.

2.1 GENERALIDADES DE LA INVESTIGACIÓN

La presente investigación se estructuró como un estudio de tipo aplicado, dado que se orientó a la resolución de un problema logístico específico mediante la implementación de un prototipo tecnológico. El proyecto trasladó los fundamentos teóricos del *Edge Computing* y la visión artificial hacia una solución práctica que automatiza el conteo de pasajeros en la ruta Bolívar - El Ángel, gestionada por la operadora TAXIBOCARCHI. De esta manera, el trabajo se enfocó en generar una herramienta tecnológica viable para optimizar los procesos de despacho vehicular y reducir los recorridos ineficientes.

Enfoque de la investigación

El enfoque metodológico adoptado fue de carácter cuantitativo. Se recolectaron y evaluaron métricas de rendimiento del *hardware* y *software*, tales como la precisión del algoritmo de detección, la latencia de procesamiento de la placa ESP32-S3 y la tasa de inferencia en fotogramas por segundo. Esta perspectiva cuantitativa permitió validar la robustez técnica del dispositivo mediante criterios de rendimiento objetivos y verificables,

confrontando los indicadores medidos con los umbrales de aceptación definidos en los requisitos no funcionales.

2.2 TECNICAS DE RECOLECCIÓN DE DATOS

Para diagnosticar la situación inicial y definir los requerimientos técnicos del sistema automatizado, se emplearon técnicas de recolección de información primaria directamente en el entorno de estudio. Estas herramientas metodológicas fueron fundamentales para comprender las restricciones físicas del lugar y el comportamiento de los usuarios, elementos críticos para el diseño la arquitectura. La herramienta seleccionada fue la observación de campo directa, la cual permitió documentar empíricamente las brechas operativas del servicio, identificar las fallas logísticas de la compañía TAXIBOCARCHI y fundamentar la viabilidad tecnológica del dispositivo a implementar.

2.2.1 Instrumentos de recolección de datos

2.2.1.1 Observación directa

La observación directa se ejecutó con el propósito de examinar la dinámica operativa real del modelo de transporte a demanda. Para obtener un diagnóstico integral, esta técnica se dividió en dos escenarios de estudio complementarios: el punto de convergencia físico (la parada) donde inicia el servicio, y la trayectoria vial (la ruta) que recorren las unidades vehiculares. La recolección de esta información visual permitió evidenciar los tiempos de espera irregulares y la intermitencia de conectividad, factores que justificaron el procesamiento local de las imágenes en el microcontrolador.

Los datos recabados durante las jornadas de campo se sistematizaron mediante instrumentos de registro estructurados. La Tabla 1 detalla la observación realizada en el punto principal de congregación de pasajeros.

Tabla 1. *Matriz de observación de la dinámica operativa en la zona de embarque (Parada)*

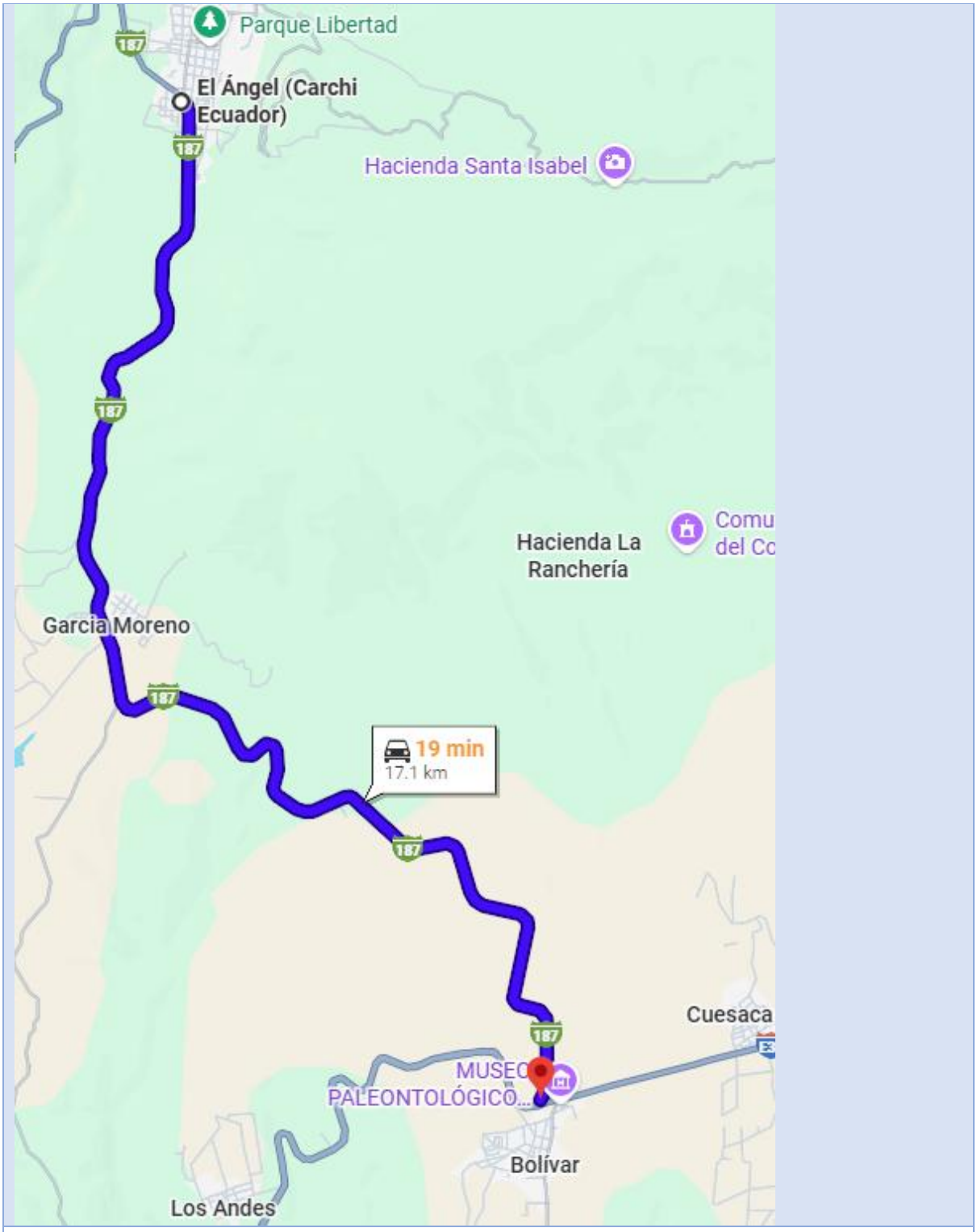
FICHA DE OBSERVACIÓN
Fecha: 12 de enero de 2026
Lugar: Parada principal de la ruta Bolívar - El Ángel.
Descripción: Proceso de aglomeración de pasajeros y evaluación empírica del aforo para el despacho vehicular.

Nota. Fotografía del punto físico de convergencia de usuarios. En la imagen se evidencia el escenario operativo tradicional, destacando la carencia de infraestructura tecnológica para el monitoreo de la demanda. Se observa cómo los pasajeros esperan a la intemperie hasta alcanzar el volumen necesario para que el conductor decida despachar la unidad, validando la necesidad de automatizar el área.

Complementariamente, se realizó una inspección del trayecto físico para evaluar las condiciones externas a las que se enfrenta la flota vehicular y el sistema de comunicaciones. El detalle de esta observación se expone en la Tabla 2.

Tabla 2. ***Registro de observación de las condiciones viales y de conectividad (Ruta)***

FICHA DE OBSERVACIÓN
Fecha: 15 de enero de 2026
Lugar: Trayectoria vial del eje Bolívar - El Ángel.
Descripción: Verificación de recorridos sin demanda e inspección de las zonas de cobertura celular.



Nota. Fotografía de un tramo del trayecto rural recorrido por las unidades de TAXIBOCARCHI. La observación de esta ruta permitió documentar la incidencia de movilizar unidades vacías y constatar las áreas geográficas que presentan intermitencia en la conectividad móvil.

Para el desarrollo del presente proyecto se adoptó un enfoque metodológico híbrido organizado en dos fases claramente diferenciadas. La Fase 1 abarcó el desarrollo del prototipo de hardware e inteligencia artificial bajo el Modelo de Prototipos de Pressman, y la Fase 2

comprendió el desarrollo de la interfaz web bajo la metodología ágil Scrum. Esta división respondió a la naturaleza diferenciada de cada componente del sistema. El Modelo de Prototipos de Pressman fue seleccionado para la Fase 1 por su capacidad para responder a los retos específicos de integrar hardware de bajo consumo con modelos de inteligencia artificial en tiempo real, destacando por su flexibilidad y adaptabilidad en entornos de *Edge Computing*. Para la Fase 2, correspondiente a la visualización y almacenamiento de datos, se optó por la metodología ágil Scrum por su idoneidad para el desarrollo iterativo de la interfaz web.

El propósito del presente proyecto fue facilitar el análisis, diseño y evaluación inicial de un sistema inteligente de conteo de pasajeros personalizado para la ruta Bolívar - El Ángel de la compañía de transporte TAXIBOCARCHI. A través del desarrollo de este prototipo, se buscó validar la viabilidad técnica del sistema, observar su comportamiento en la detección de usuarios e identificar mejoras operativas antes de una posible implementación final. Este enfoque permitió responder al objetivo de resolver las brechas logísticas de la operadora, sentando las bases comprobadas para el desarrollo futuro de un dispositivo tecnológico que cumpla con los requerimientos específicos del entorno rural y las necesidades de los conductores.

2.3. Alcance

El alcance de este proyecto se delimitó estrictamente al diseño, desarrollo y validación de un prototipo funcional que demostró la viabilidad tecnológica del sistema automatizado para la detección de pasajeros. El prototipo cubrió el ciclo demostrativo completo: captura de información visual mediante el sensor OV2640, procesamiento local de los datos a través del algoritmo FOMO ejecutado en el microcontrolador ESP32-S3, envío de notificaciones en tiempo real a la plataforma de mensajería Telegram, y visualización del estado del conteo mediante un panel web accesible desde el navegador.

Aunque el proyecto se fundamentó y contextualizó en las necesidades logísticas de la ruta Bolívar - El Ángel operada por la compañía TAXIBOCARCHI, la validación del dispositivo se realizó de manera exclusiva en un entorno de laboratorio controlado que simuló las condiciones de dicha parada de transporte.

En consecuencia, al tratarse netamente de un prototipo de investigación, quedó totalmente fuera del alcance la instalación física del equipo a la intemperie, el diseño de blindaje o certificaciones industriales contra el clima, la integración definitiva en la infraestructura de la operadora y su operación ininterrumpida a largo plazo en un entorno real de producción. El resultado de este trabajo se entregó como una prueba de concepto tecnológica compuesta por dos entregables: el nodo IoT de detección por visión artificial y el panel web de monitoreo desarrollado bajo metodología Scrum, ambos integrados en una solución funcional para la compañía TAXIBOCARCHI.

2.4. Fase 1: Metodología de desarrollo del prototipo

El desarrollo del sistema se estructuró siguiendo el modelo de prototipos, una metodología que permitió la construcción de versiones funcionales y evaluables antes de consolidar el diseño definitivo (Figura 4). Este enfoque facilitó la integración del hardware, los algoritmos de visión artificial y las interfaces gráficas mediante ciclos de mejora continua.

Estas versiones funcionales tempranas consistieron en la construcción del nodo de percepción, integrando el microcontrolador ESP32-S3 y el módulo de cámara. Esta metodología permitió evaluar el desempeño del algoritmo de detección FOMO de forma continua. A través del ciclo de prototipado se facilitó la identificación temprana de necesidades técnicas y cuellos de botella, tales como limitaciones de memoria, ajustes de iluminación para el sensor óptico y la latencia en la transmisión del formato JSON a través de la API de Telegram. Las iteraciones constantes permitieron que el hardware y el software se adaptaran a las condiciones reales de las paradas en la ruta Bolívar - El Ángel, refinando el dispositivo tras

cada prueba hasta alcanzar un diseño estable, preciso y operativo para la compañía TAXIBOCARCHI. Las fases que componen esta metodología se detallan gráficamente en la Figura 4.

Figura 4. *Ciclo de vida del desarrollo por prototipos*

Nota. La imagen representa el modelo iterativo utilizado en la construcción del sistema, el cual sugiere una repetición continua de las fases de diseño y evaluación hasta consolidar el producto de ingeniería final. Obtenida de *https://is3ados.blogspot.com/2008/11/los-ciclos-de-vida-del-software-ii.html*.

2.4.1. Análisis y refinamiento de requisitos

En esta etapa inicial, se procesaron los hallazgos obtenidos en el trabajo de campo con la compañía TAXIBOCARCHI para definir las especificaciones técnicas del sistema. El análisis se centró en identificar las funciones necesarias para resolver el problema de conteo de pasajeros y los atributos de calidad requeridos para que la solución fuera fiable en un entorno real.

2.4.1.1. Requisitos funcionales

Los requisitos funcionales (Tabla 3) determinaron las capacidades operativas del sistema, incluyendo la captura de datos, el procesamiento en el borde y la visualización de la información tanto en dispositivos móviles como en una interfaz de escritorio para monitoreo centralizado.

Tabla 3. *Requisitos funcionales del sistema IoT y visión artificial*

REQUISITOS FUNCIONALES	
Código	**Descripción del Requisito Funcional**
RF-01	El prototipo debe capturar imágenes del entorno mediante el sensor OV2640 de forma automática.
RF-02	El prototipo debe procesar las imágenes localmente utilizando la arquitectura de red neuronal FOMO para garantizar la privacidad.
RF-03	El prototipo debe cuantificar la presencia de pasajeros en el punto de parada en tiempo real.
RF-04	El prototipo debe transmitir las alertas de aforo hacia la plataforma Telegram para la notificación inmediata al conductor.
RF-05	El prototipo debe estar integrado un panel de visualización web que permita verificar el estado del conteo y los datos históricos desde un navegador.
RF-06	El prototipo debe empaquetar los datos en formato JSON para asegurar la compatibilidad entre el hardware y las interfaces gráficas.

Nota. Funciones lógicas implementadas para el funcionamiento integral del prototipo y su monitoreo remoto.

2.4.1.2. Requisitos no funcionales

Los requisitos no funcionales (Tabla 4) definieron los estándares de calidad y rendimiento que el sistema debió cumplir para ser considerado exitoso. Se puso especial énfasis en la precisión del algoritmo y la accesibilidad de la interfaz gráfica desarrollada.

Tabla 4. *Requisitos no funcionales y atributos de calidad*

REQUISITOS NO FUNCIONALES	
Atributo	**Descripción del Requisito No Funcional**
Privacidad	El prototipo debe procesar las imágenes en la memoria local del microcontrolador, sin transmitir ni almacenar ninguna fotografía de forma externa.
Seguridad	El prototipo debe transmitir los datos hacia la API de Telegram mediante el protocolo HTTPS, garantizando el cifrado de extremo a extremo de cada paquete JSON enviado.
Precisión	El prototipo debe alcanzar un nivel de exactitud superior al 85% en la detección de usuarios en condiciones de iluminación variable.
Rendimiento	El prototipo debe garantizar que el tiempo de respuesta entre la captura del sensor y la actualización del panel web no supere los 3 segundos.
Usabilidad	El panel web debe permitir que un usuario sin formación técnica complete las tareas principales de monitoreo en menos de 2 minutos, validado mediante prueba con al menos 3 usuarios.
Compatibilidad	El panel de visualización debe ser accesible desde navegadores web estándar para facilitar su uso en cualquier equipo de cómputo.

Nota. Atributos de calidad y rendimiento utilizados para auditar la efectividad del prototipo tecnológico.

2.4.1.3. Historias de Usuario

Las historias de usuario permiten traducir las necesidades reales de los actores involucrados en funcionalidades técnicas concretas, asegurando que el sistema no solo sea viable desde el punto de vista de la ingeniería, sino también pertinente para quienes lo utilizarán en su operación diaria. En este proyecto, las historias fueron levantadas a partir de los hallazgos del trabajo de campo y la observación directa del funcionamiento operativo de TAXIBOCARCHI, abarcando las perspectivas del conductor y el administrador de flota.

El conjunto de historias corresponde exclusivamente al componente IoT del sistema, es decir, al nodo de detección basado en el microcontrolador ESP32-S3. Las historias de usuario del panel web de monitoreo, desarrollado bajo metodología Scrum, se documentan de forma independiente en la sección 2.5.5. Esta separación respeta la naturaleza diferenciada de cada componente y la metodología aplicada en su desarrollo. La Tabla 5 detalla el conjunto de historias de usuario definidas para el componente IoT.

Tabla 5. *Historias de usuario del sistema de monitoreo y alertas*

ID	Nombre	Est. (h)	Imp.	Descripción	Criterios de aceptación	Dependencia
HU-01	Alerta definitiva de aforo	30	1000	Como conductor de la unidad, deseo recibir una notificación urgente cuando se hayan reunido 4 pasajeros en la parada, para iniciar el recorrido de forma inmediata y eficiente.	• Envío automático del mensaje al bot de Telegram al detectar ≥ 4 personas. • Tiempo de notificación inferior a 3 segundos. • El mensaje debe indicar la cantidad de pasajeros y la acción requerida.	Ninguna
HU-02	Alerta preventiva de demanda	20	900	Como conductor de la unidad, deseo recibir un aviso anticipado cuando se estén reuniendo 3	• Envío de alerta preventiva al detectar exactamente 3 personas.	HU-01

				pasajeros en la parada, para acercarme con anticipación y reducir los tiempos de espera.	• El mensaje debe diferenciarse visualmente de la alerta definitiva. • El sistema debe respetar un intervalo mínimo de 60 segundos entre alertas consecutivas para evitar saturación.	
HU-03	Detección autónoma con IA	50	950	Como administrador de la flota, deseo que el dispositivo realice el conteo sin intervención humana, para obtener datos objetivos y automatizar el despacho de unidades.	• Activación automática del sensor al encender el dispositivo. • Identificación de siluetas humanas mediante el modelo FOMO. • Ejecución local de la inferencia sin dependencia de servidores externos.	Ninguna
HU-04	Privacidad de los pasajeros	45	950	Como administrador del servicio, deseo que el sistema procese las imágenes localmente sin almacenarlas ni transmitirlas, para que los derechos a la privacidad y protección de datos de los usuarios sean respetados.	• Descarte inmediato del fotograma en RAM tras la inferencia. • Transmisión exclusiva del dato numérico en formato JSON. • Ausencia de almacenamiento físico de imágenes en el dispositivo o en la nube.	HU-03
HU-06	Resiliencia de conectividad	25	700	Como administrador del sistema, necesito que el dispositivo gestione reconexiones automáticas a la red Wi-Fi, para garantizar la continuidad de las alertas en zonas con señal inestable.	• Detección de fallos de red en el firmware. • Intentos de reconexión cíclica sin bloquear el proceso de conteo e inferencia. • Notificación del estado de conexión al iniciar el sistema.	HU-01

Nota. Historias de usuario levantadas a partir del trabajo de campo con la compañía TAXIBOCARCHI y complementadas con los requisitos técnicos del sistema. La columna Est. (h) indica la estimación de horas de desarrollo; Imp. indica el valor de importancia asignado por el equipo.

Con el fin de vincular las historias de usuario con las fases de construcción del prototipo, la Tabla 6 presenta el plan de iteraciones del desarrollo, agrupando cada historia de usuario con la fase del modelo de prototipos que le corresponde y el entregable funcional asociado.

Tabla 6. *Plan de iteraciones del prototipo vinculado a las historias de usuario*

Iteración	Historias de usuario atendidas	Entregable funcional
1	HU-03, HU-04	Hardware ensamblado + modelo FOMO entrenado en Edge Impulse
2	HU-01, HU-02, HU-06	Firmware con inferencia local + alertas Telegram con doble umbral + resiliencia Wi-Fi
3	HU-SW-01 a HU-SW-07	Panel web completo (NestJS + React + Firebase) desarrollado bajo Scrum
4	HU-01 a HU-04, HU-06 / HU-SW-01 a HU-SW-06	Integración y pruebas del sistema completo (Sección 2.6)

Nota. Distribución de las historias de usuario en iteraciones del ciclo de desarrollo del prototipo bajo el modelo de Pressman. Cada iteración produce un entregable funcional verificable.

2.4.2. Diseño rápido del prototipo

Una vez establecidos los requisitos funcionales y no funcionales, se procedió a la fase de diseño rápido. En esta etapa se elaboraron los esquemas arquitectónicos y lógicos que sirvieron como plano de ingeniería para la construcción del dispositivo. Este diseño preliminar definió la interacción exacta entre los componentes de *hardware* (nodo de procesamiento en el borde), los protocolos de red y la plataforma visual del usuario final.

2.4.2.1. Arquitectura funcional del prototipo

El diseño arquitectónico definió el modo en que el microcontrolador XIAO ESP32-S3 Sense interactuaría con su entorno físico y lógico, así como la manera en que la información recolectada sería transformada, transmitida y consumida por los actores finales del sistema. Siguiendo el modelo de arquitectura de tres capas propuesto por Kumar et al. (2024) para sistemas del Internet de las Cosas, ampliamente adoptado como estándar de facto en la literatura especializada, la solución se estructuró en tres capas funcionales independientes pero

interconectadas: la capa de percepción —denominada también capa de cómputo en el borde o *Edge* en arquitecturas modernas con procesamiento local—, la capa de red o comunicación, y la capa de aplicación e interfaz. Esta segmentación permitió aislar responsabilidades, facilitar el mantenimiento del sistema y garantizar el cumplimiento de los requisitos de privacidad y rendimiento establecidos en la Tabla 4.

Capa 1 — Computación en el borde

La capa de borde concentró toda la lógica de captura y procesamiento de información visual, ejecutándose íntegramente dentro del microcontrolador XIAO ESP32-S3 Sense sin dependencia de recursos computacionales externos. Sus componentes funcionales fueron los siguientes:

- **Sensor óptico:** el módulo OV2640 capturó fotogramas a una resolución de 96×96 píxeles en escala de grises, valor seleccionado como balance óptimo entre la calidad de información necesaria para la detección de siluetas humanas y la carga computacional admisible por la SRAM del microcontrolador.
- **Motor de inferencia:** el modelo FOMO, exportado desde Edge Impulse en formato de librería Arduino C++, fue desplegado directamente sobre la memoria del ESP32-S3 con cuantización int8. La inferencia se ejecutó en local, devolviendo el conteo de detecciones de la clase hum por cada fotograma procesado, con un tiempo promedio de 459 ms por inferencia.
- **Lógica de descarte y privacidad:** tras cada inferencia, el fotograma original fue eliminado inmediatamente de la memoria RAM, garantizando que ninguna imagen fotográfica fuera transmitida ni almacenada externamente. Únicamente el dato numérico resultante (entero correspondiente al conteo) avanzó hacia la siguiente capa.

Esta arquitectura edge-first fue determinante para el cumplimiento del requisito no funcional de privacidad, ya que eliminó por diseño la posibilidad de fuga de información biométrica o visual hacia la red.

Capa 2 — Comunicación

La capa de comunicación se encargó del transporte seguro y eficiente del dato numérico desde el dispositivo de borde hacia los servicios de procesamiento centralizado. Para esta función se optó por un esquema basado en el protocolo HTTP/HTTPS con arquitectura REST, en lugar de protocolos de mensajería ligera como MQTT, por dos razones técnicas concretas: la integración nativa con la API de Telegram, que opera exclusivamente sobre HTTPS, y la simplicidad operativa que ofrece al no requerir el despliegue de un broker intermedio. La selección de HTTPS como protocolo de transporte resultó además la opción técnicamente más adecuada para el contexto del sistema, dado que su compatibilidad universal con clientes HTTP estándar y el cifrado TLS integrado garantizan la seguridad de las comunicaciones sin complejidad operativa adicional.

Los componentes funcionales de esta capa fueron:

- **Módulo de transporte:** el módulo Wi-Fi integrado del ESP32-S3 (802.11 b/g/n a 2.4 GHz) estableció la conexión con la red local, con reconexión cíclica automática gestionada por el firmware en caso de pérdida de señal.
- **Empaquetado de datos:** el conteo resultante fue serializado en formato JSON con la estructura {"personas": <int>}, optimizando el tamaño del payload a unos pocos bytes por transmisión.
- **Protocolo de transporte seguro:** las peticiones se enviaron mediante método POST sobre HTTPS hacia el endpoint del servidor backend, garantizando el cifrado de extremo a extremo de cada paquete y cumpliendo con el requisito no funcional de seguridad.

Capa 3 — Aplicación e interfaz

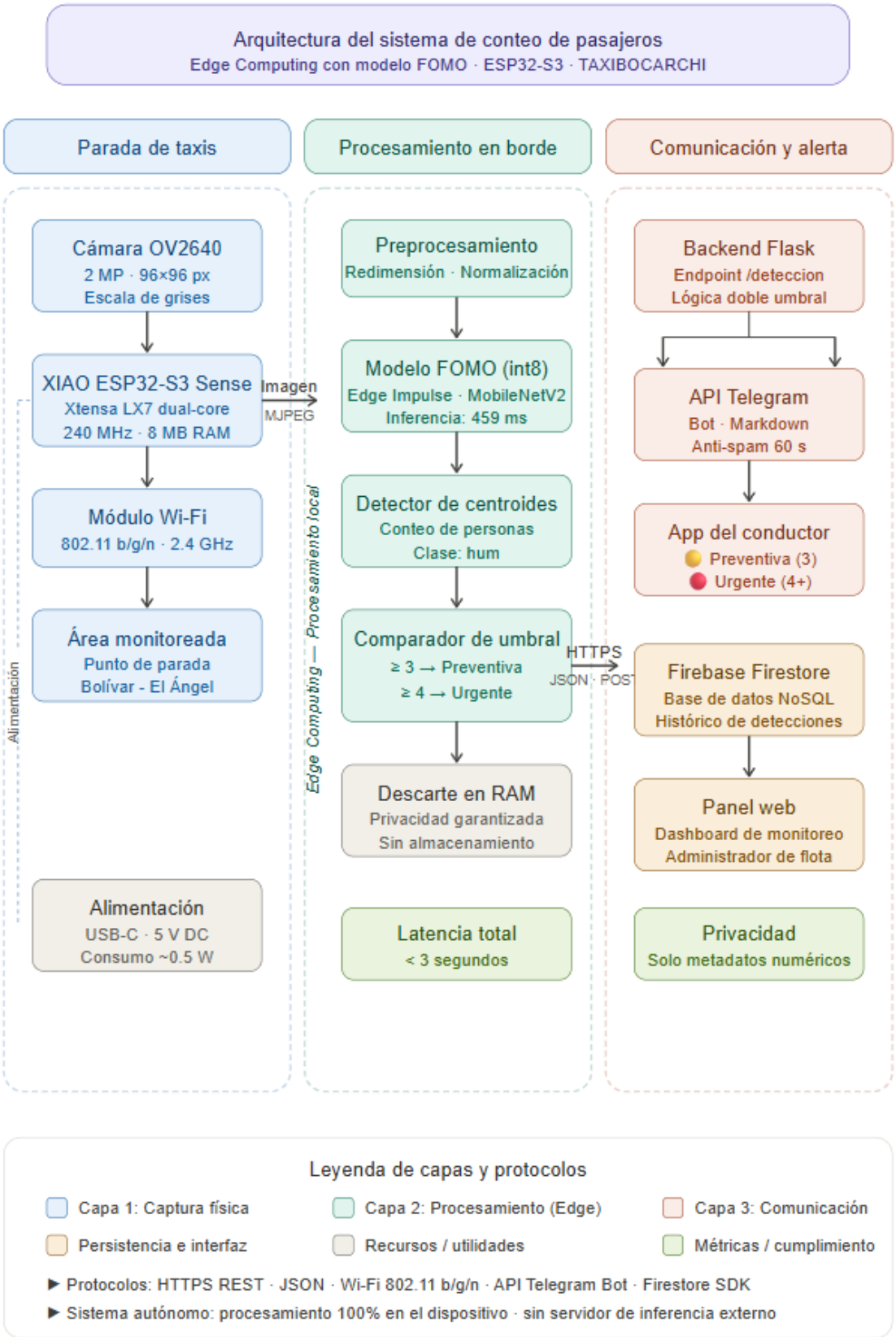
La capa de aplicación centralizó la lógica de negocio y la presentación de los resultados a los usuarios finales. A diferencia de arquitecturas más simples donde el dispositivo de borde notifica directamente al usuario, en este sistema se introdujo un servidor backend intermedio que permitió aplicar reglas de negocio complejas, persistir el historial de eventos y desacoplar el dispositivo de los canales de notificación.

Los componentes funcionales de esta capa fueron:

- **Backend de procesamiento:** un servidor Python desarrollado con el framework Flask recibió las peticiones del ESP32-S3 en el endpoint /deteccion, aplicó la lógica de doble umbral (alerta preventiva al detectar 3 personas, alerta definitiva al detectar 4 o más) y gestionó el mecanismo anti-spam con un intervalo mínimo de 60 segundos entre notificaciones consecutivas.
- **Persistencia de datos:** cada evento procesado fue almacenado en Firebase Firestore con marca temporal, conteo registrado y estado de alerta asociado, conformando el histórico operativo del sistema y la fuente de datos para análisis posterior.
- Canal de notificación al conductor: la API de Telegram operó como interfaz de alerta inmediata, entregando mensajes formateados en Markdown con codificación cromática (Amarilla es preventiva, Roja es definitiva) directamente al dispositivo móvil del conductor.
- **Panel de monitoreo web:** una interfaz web accesible desde navegador estándar consumió los datos almacenados en Firestore, permitiendo al administrador de flota visualizar el estado actual del conteo y consultar el historial de detecciones por períodos.

La interacción detallada entre estos componentes y el flujo de información completo se ilustra en la Figura 5.

Figura 5. *Arquitectura funcional del sistema — distribución en tres capas*



Nota. Representación de la arquitectura de tres capas: percepción (ESP32-S3 + FOMO), comunicación (HTTP/HTTPS) y aplicación (Firebase + Telegram + panel web).

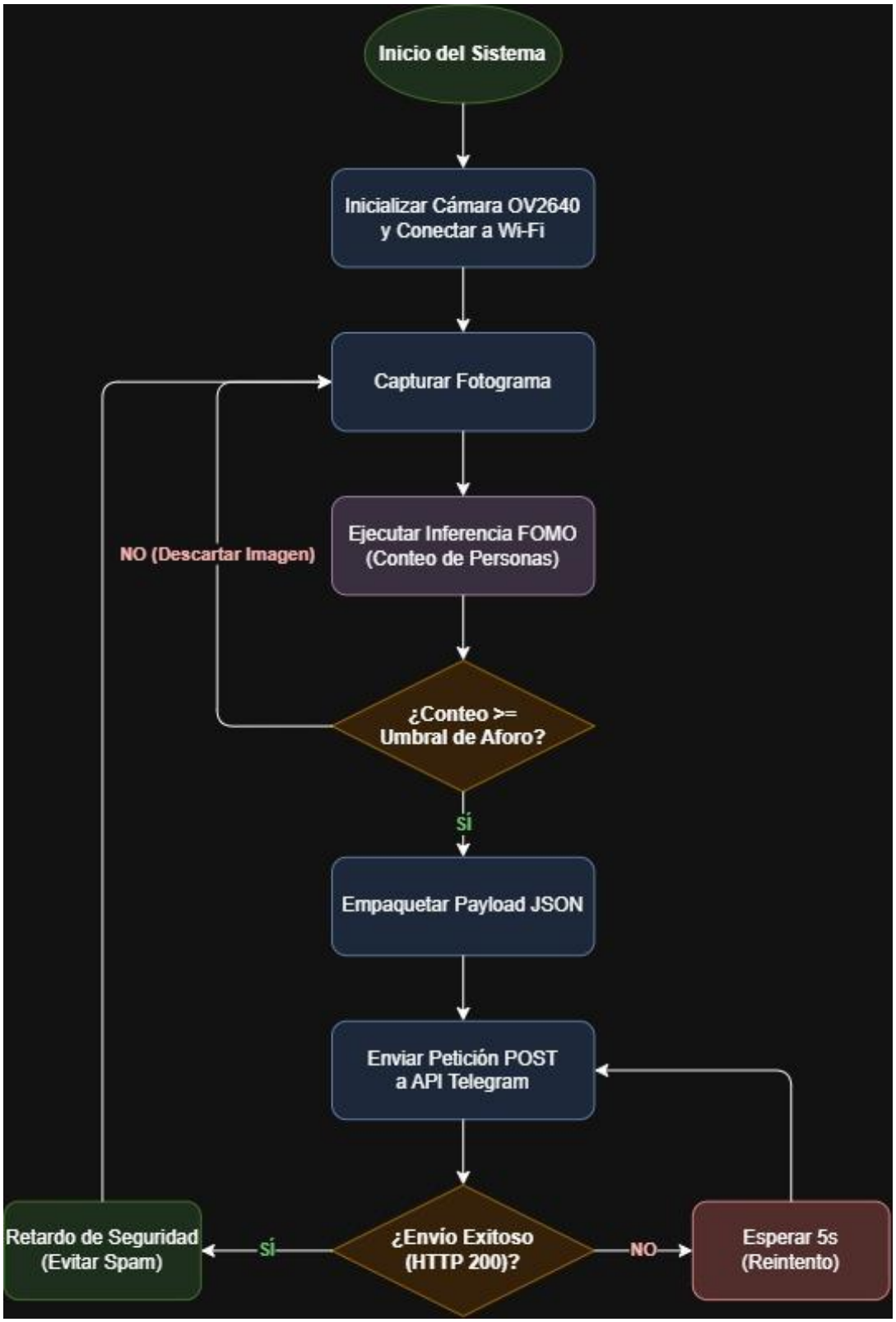
2.4.2.1. Flujograma lógico del *firmware*

Previo al diseño de *hardware*, se estructuró el flujo lógico que gobernaría el *firmware* programado en el microcontrolador. Este algoritmo definió la secuencia de operaciones paso a paso, garantizando que el dispositivo operara de manera autónoma y tolerante a fallos de red.

El ciclo lógico iniciaba con el arranque del sistema y la configuración de los pines de la cámara, seguido por el intento de conexión a la red Wi-Fi. Una vez establecido el enlace, el dispositivo entraba en un bucle continuo de evaluación: capturaba un fotograma, aplicaba los pesos de la red neuronal (*TinyML*) y generaba un conteo. El código incluía una estructura de decisión principal; si el número de pasajeros detectados era igual o mayor al límite preestablecido, se construía el mensaje JSON y se invocaba a la API de Telegram. Si el aforo no era suficiente, la imagen se descartaba inmediatamente y el ciclo se reiniciaba. Adicionalmente, se incluyeron rutinas de manejo de errores para reconectar el módulo Wi-Fi en caso de pérdida de señal, asegurando la continuidad del monitoreo.

El comportamiento secuencial de este algoritmo se representa en la Figura 6.

Figura 6. *Flujograma algorítmico del firmware integrado*



Nota. Estructura lógica del código implementado en el entorno Arduino IDE. El diagrama detalla el proceso cíclico de captura, inferencia condicional y transmisión de alertas, incluyendo los mecanismos de descarte de imágenes para protección de la privacidad.

2.4.3. Construcción e implementación técnica

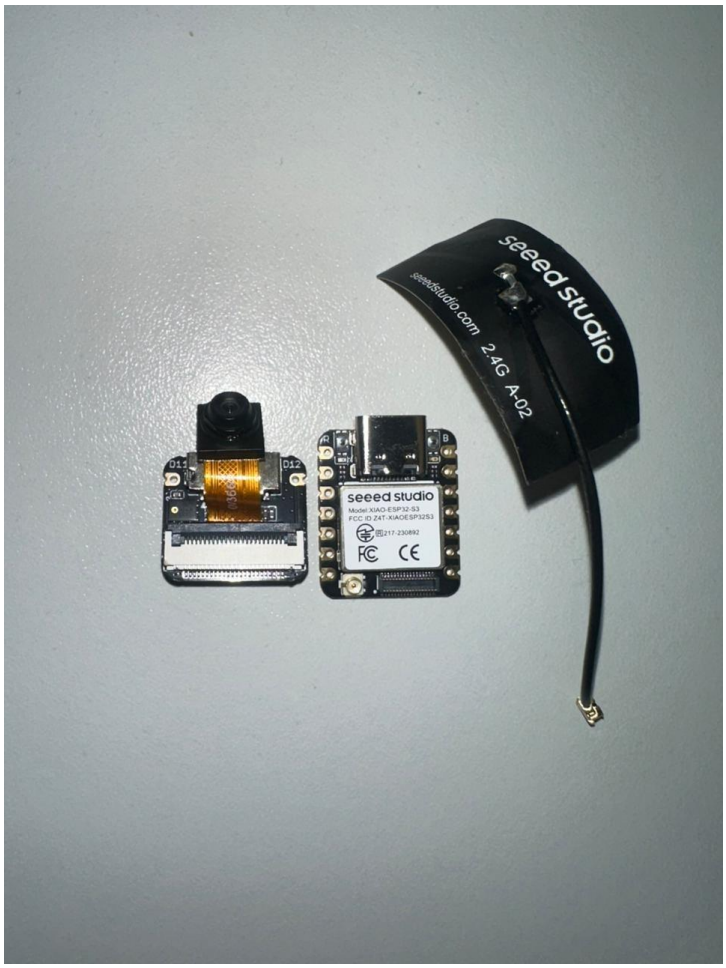
2.4.3.1. Entorno tecnológico de hardware

Esta sección comprendió el ensamblaje físico del nodo de hardware, integrando los componentes seleccionados durante la etapa de diseño. El núcleo del prototipo se construyó sobre el microcontrolador XIAO ESP32-S3 Sense, elegido por su arquitectura de doble núcleo Xtensa LX7 a 240 MHz, su módulo Wi-Fi 802.11b/g/n integrado y su compatibilidad nativa con frameworks de *TinyML*. La selección de este componente respondió directamente al requisito no funcional de procesamiento local, dado que su unidad de punto flotante (FPU) permite ejecutar operaciones de inferencia neuronal sin depender de recursos externos.

El sensor óptico utilizado fue la cámara OV2640, ensamblada directamente sobre el conector FPC del microcontrolador. Este módulo opera a una resolución configurable de hasta 1600×1200 píxeles, aunque para la inferencia del modelo FOMO se restringió a una resolución de 96×96 píxeles en escala de grises, reduciendo significativamente la carga computacional sobre el procesador sin comprometer la capacidad de detección de siluetas humanas. La alimentación del conjunto se gestionó a través del puerto USB-C del microcontrolador, garantizando una tensión estable de 3.3 V para todos los periféricos integrados.

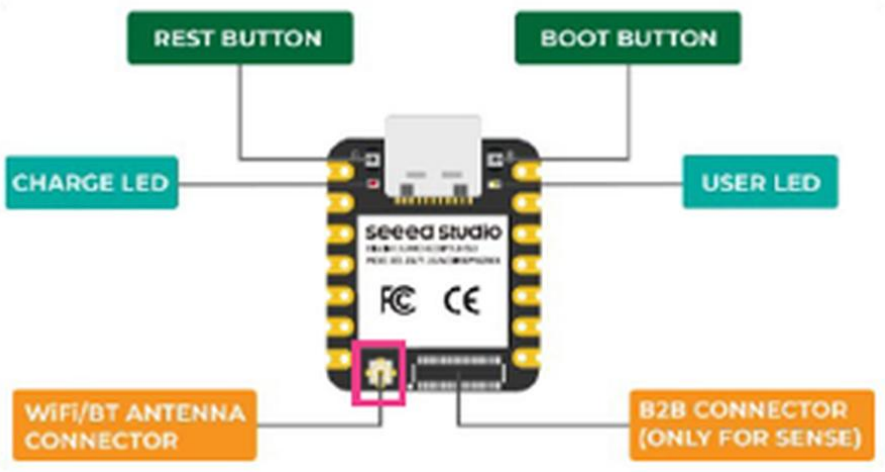
El ensamblaje no requirió soldadura adicional ni circuitos externos, dado que el módulo de cámara forma parte del diseño integrado de la placa. La configuración final del prototipo resultó en un nodo compacto y autónomo, cuyas dimensiones reducidas permitieron su instalación discreta en el entorno de prueba de laboratorio. La disposición física de los componentes empleados en el ensamblaje se ilustra en la Figura 7, mientras que la Figura 8 detalla la identificación de pines y conectores del microcontrolador XIAO ESP32-S3 (Seeed Studio, 2024).

Figura 7. *Componentes físicos del nodo de hardware del sistema TaxiBocarchi*



Nota. Componentes empleados en el ensamblaje del nodo de detección: el microcontrolador Seeed Studio XIAO ESP32-S3 Sense, el módulo de cámara OV2640 conectado mediante conector FPC y la antena Wi-Fi/Bluetooth externa de 2.4 GHz.

Figura 8. *Identificación de pines y conectores del XIAO ESP32-S3*



Nota. Adaptado de Seeed Studio (2024). Recuperado de https://files.seeedstudio.com/wiki/SeeedStudio-XIAO-ESP32S3/img/front-indication.png

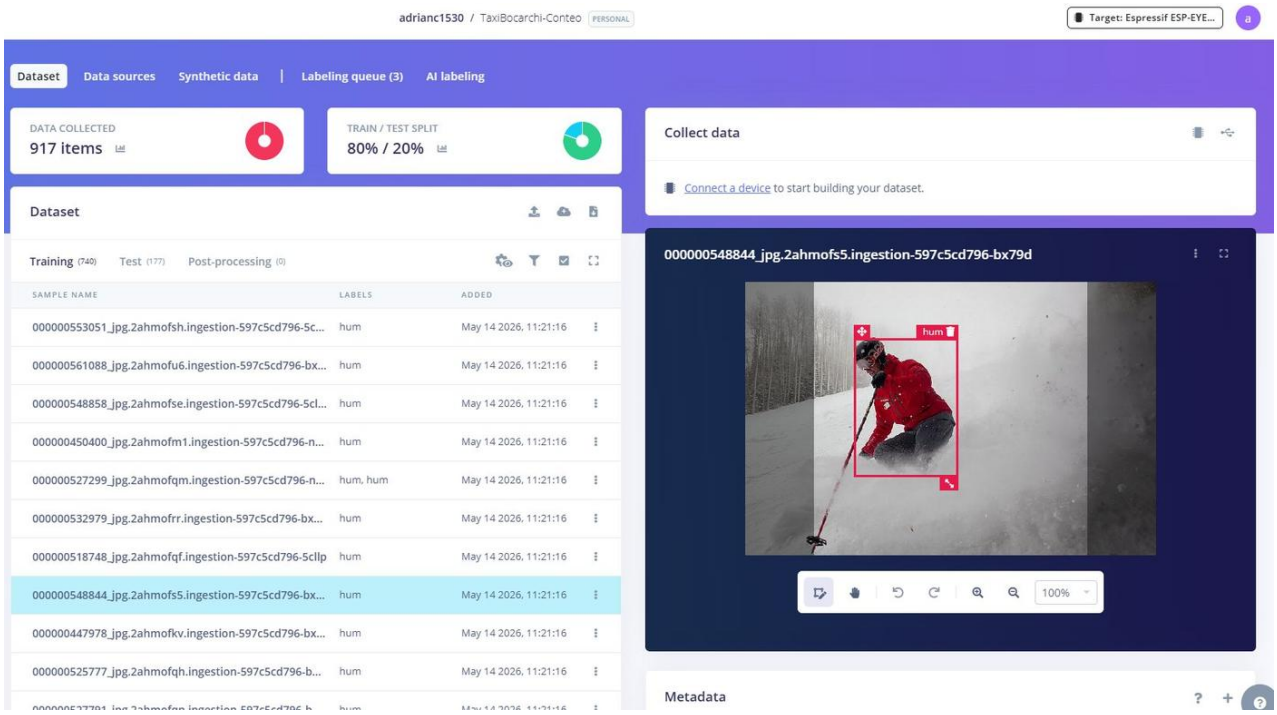
2.4.3.2. Entorno tecnológico de Machine Learning

Para dotar al prototipo de capacidad de detección autónoma, la construcción del modelo de visión artificial se ejecutó íntegramente en la plataforma Edge Impulse, herramienta especializada en el desarrollo de modelos de TinyML orientados a dispositivos embebidos con recursos computacionales restringidos.

2.4.3.2.1 Ingesta y partición del dataset

El conjunto de datos se obtuvo a partir de un dataset público de detección de personas disponible en la biblioteca abierta de la plataforma Edge Impulse, etiquetado bajo las clases *hum* (presencia de persona) y *background* (fondo sin persona). El dataset empleado corresponde al proyecto de acceso abierto Person Detection FOMO de Edge Impulse, cuya distribución de clases se presenta en la Figura 9 (Edge Impulse Inc., 2023), disponible en la biblioteca pública de la plataforma. Este conjunto fue seleccionado por contener imágenes representativas de personas en diversos ángulos y condiciones de iluminación, adecuadas para el entrenamiento del modelo de detección. El dataset consolidado alcanzó un total de 917 muestras, divididas automáticamente bajo una proporción estándar de 80% para entrenamiento y 20% para validación, resultando en aproximadamente 733 imágenes de entrenamiento y 184 de validación. En cuanto a su composición, las escenas presentan entre una y cinco personas por imagen, capturadas a distancias aproximadas de entre dos y cinco metros respecto de la cámara y con una variabilidad deliberada tanto de ángulos (tomas frontales, laterales y lejanas) como de condiciones de iluminación. Esta heterogeneidad fue determinante en la selección del conjunto, ya que reproduce las situaciones que se presentan en una parada de taxis real y favorece la capacidad de generalización del modelo ante escenarios no vistos durante el entrenamiento.

Figura 9. *Vista de adquisición de datos en Edge Impulse — dataset TaxiBocarchi-Conteo*



Nota. Captura de la plataforma Edge Impulse mostrando el dataset consolidado de 917 muestras con la partición 80%/20% para entrenamiento y validación. Se observan las muestras etiquetadas bajo la clase hum con sus respectivos bounding boxes.

2.4.3.2.2 Configuración del impulso y preprocesamiento

Dentro de la sección *Impulse Design*, se definió una capa de entrada de 12.288 características, correspondiente a imágenes de 96×96 píxeles en escala de grises redimensionadas automáticamente por el pipeline de preprocesamiento. La arquitectura neuronal seleccionada fue FOMO (Faster Objects, More Objects), un modelo de detección de objetos ultraligero diseñado específicamente para microcontroladores, cuya implementación en Edge Impulse opera sobre una base MobileNetV2 simplificada.

2.4.3.2.3 Hiperparámetros de entrenamiento

Los parámetros configurados en la sección *Neural Network Settings* para el proceso de entrenamiento fueron los siguientes:

- Ciclos de entrenamiento (épocas): 300
- Tasa de aprendizaje: 0.015

- Tamaño de lote (*batch size*): 32
- Tamaño del conjunto de validación: 20%
- Cuantización del modelo: int8 (activada mediante *Profile int8 model*)
- Procesador de entrenamiento: CPU
- Optimizador aprendido: desactivado
- Aumento de datos (*data augmentation*): activado

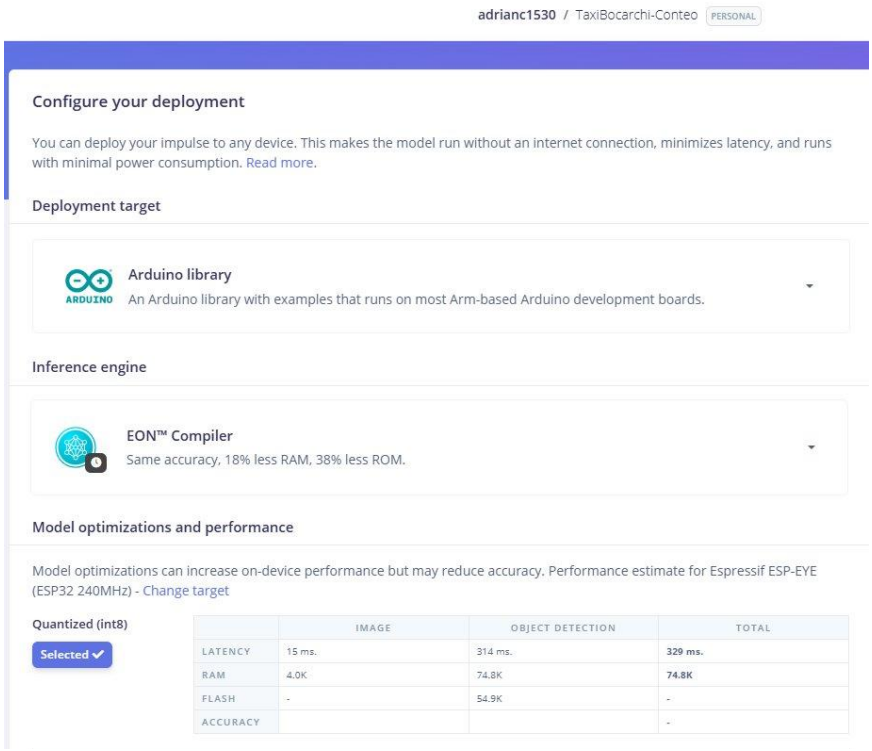
La matriz de confusión evidenció una capacidad de discriminación muy alta para el fondo vacío y una mejora sustancial en la detección de personas respecto a iteraciones anteriores del prototipo

Los resultados del entrenamiento del modelo FOMO, incluyendo las métricas de F1 Score, precisión, recall y la matriz de confusión, se presentan en el Capítulo III (sección 3.1.1), junto con el análisis de su significado en el contexto del sistema TaxiBocarchi.

- Uso pico de RAM: **69.4 KB**
- Uso de memoria Flash: **68.9 KB**

La Figura 10 resume la configuración final del despliegue exportado a Arduino IDE. Asimismo, la Tabla 7 sintetiza los hiperparámetros de entrenamiento y las métricas de rendimiento obtenidas.

Figura 10. *Configuración del despliegue del modelo en Edge Impulse — exportación como librería Arduino*



Nota. Pantalla de despliegue de Edge Impulse mostrando la selección de Arduino library como destino de exportación y el compilador EON para optimización de RAM. La tabla de rendimiento confirma una latencia total de 329 ms y un uso de RAM de 74.8 KB en el microcontrolador objetivo (Espressif ESP32 240 MHz).

Tabla 7. *Parámetros de entrenamiento y métricas del modelo FOMO*

Parámetro / Métrica	Valor
Total de muestras	917
División entrenamiento / validación	80% / 20%
Épocas	300
Tasa de aprendizaje	0.015
Tamaño de lote	32
Cuantización	int8
F1 Score (validación)	88.3%
Precisión clase *background*	100%
Precisión clase *hum*	82.4%
Tiempo de inferencia en dispositivo	314 ms
RAM pico	69.4 KB
Flash	68.9 KB

Nota. Configuración de hiperparámetros y métricas de rendimiento del modelo FOMO entrenado en Edge Impulse para detección de pasajeros en parada de bus.

2.4.3.3. Programación del firmware y backend del sistema

Una vez validado el modelo en Edge Impulse, el siguiente paso fue trasladar toda la lógica de detección, comunicación y almacenamiento a código ejecutable. El desarrollo se dividió en dos capas bien diferenciadas: el firmware del microcontrolador, responsable de la captura e inferencia, y un servidor backend en Python que centralizó la lógica de alertas, el control anti-spam y el registro en la nube.

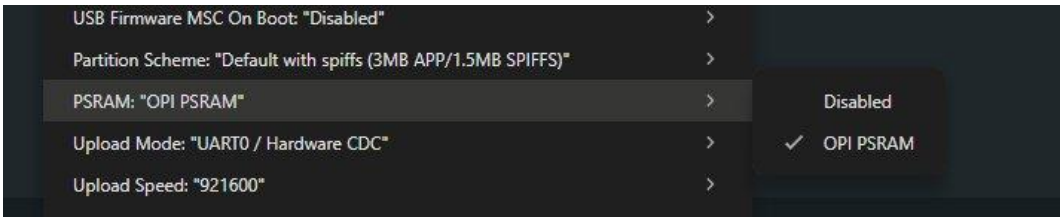
2.4.3.3.1. Integración de la librería neuronal en Arduino IDE

El modelo FOMO entrenado en Edge Impulse fue exportado como librería Arduino en formato `.zip`, la cual encapsula los pesos cuantizados int8, el pipeline de preprocesamiento y las rutinas de inferencia optimizadas para la arquitectura ESP32. La instalación se realizó desde *Sketch → Include Library → Add .ZIP Library*, exponiendo la interfaz del SDK de Edge Impulse que abstrae el motor TensorFlow Lite Micro subyacente. Esto permitió al firmware capturar un fotograma, ejecutar la inferencia y obtener el conteo de detecciones de la clase *hum* mediante llamadas a funciones estándar, sin necesidad de gestionar manualmente las operaciones matriciales del modelo.

2.4.3.3.2. Configuración del entorno y validación de la cámara

La correcta ejecución del modelo en el XIAO ESP32-S3 Sense dependió de una configuración precisa del entorno de compilación en el Arduino IDE. El parámetro más crítico fue la habilitación de la memoria PSRAM en modo OPI, indispensable para reservar el buffer de fotogramas de la cámara y los tensores de inferencia del modelo FOMO; sin esta memoria externa, el microcontrolador no dispone de suficiente RAM interna para alojar simultáneamente la imagen capturada y la red neuronal, provocando fallos de asignación de memoria en tiempo de ejecución. La Figura 11 muestra la configuración de PSRAM empleada.

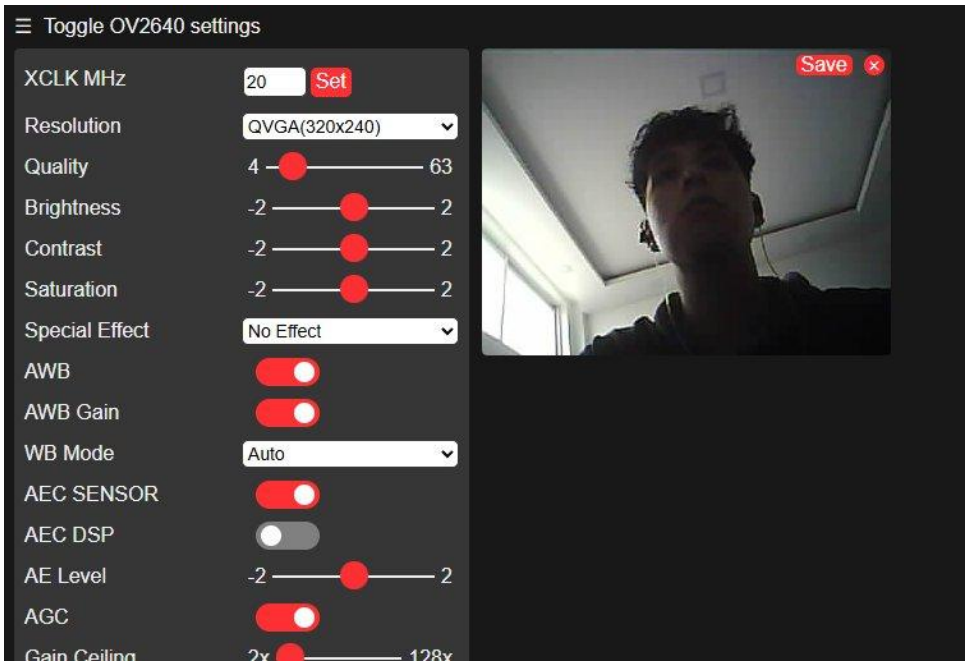
Figura 11. *Configuración de PSRAM en modo OPI en el Arduino IDE*



Nota. Selección obligatoria de la opción OPI PSRAM en la configuración de la placa. Esta memoria externa de 8 MB permite alojar el buffer de la cámara y los tensores del modelo FOMO, evitando desbordamientos de la RAM interna del ESP32-S3.

Una vez configurado el entorno, se realizaron pruebas de captura para validar el correcto funcionamiento del sensor OV2640 y ajustar los parámetros de imagen. Mediante la interfaz de configuración de la cámara se verificó la resolución QVGA (320x240), la frecuencia de reloj XCLK de 20 MHz y los ajustes de brillo, contraste y saturación, confirmando que el flujo de video se capturaba de forma estable y nítida para alimentar al modelo de inferencia. La Figura 12 evidencia una prueba de captura en vivo del sensor.

Figura 12. *Prueba de captura en vivo del sensor OV2640 del XIAO ESP32-S3 Sense*



Nota. Validación del funcionamiento de la cámara mediante la interfaz de configuración del sensor OV2640, mostrando la resolución QVGA, la frecuencia XCLK de 20 MHz y los

parámetros de imagen. La captura confirma la operatividad del sensor para la detección de personas.

2.4.3.3.3. Arquitectura del backend y lógica de umbrales

El microcontrolador no procesa la lógica de alertas de forma aislada; en su lugar, transmite el resultado de cada inferencia como un dato numérico hacia un servidor backend desarrollado en Python con el framework Flask. Este servidor, accesible mediante el endpoint `/deteccion`, recibe el conteo de personas en formato JSON y aplica la lógica de decisión centralizada.

El sistema implementó un esquema de doble umbral para gestionar las notificaciones:

- **Umbral preventivo (3 personas):** cuando el conteo alcanza las 3 personas, el sistema emite una alerta amarilla al conductor con el mensaje *"Se está formando un grupo. ¡Acércate a la parada!"*, anticipando la demanda antes de que el cupo esté completo.
- **Umbral definitivo (4 personas):** al superar las 4 personas, se activa una alerta roja con el mensaje *"¡Llenando cupo! Se necesita un taxi de inmediato."*, indicando que el aforo mínimo de despacho ha sido alcanzado.

Adicionalmente, el backend incorporó un mecanismo anti-spam con un tiempo de espera de 60 segundos entre alertas consecutivas del mismo tipo, evitando la saturación del canal de Telegram ante detecciones repetidas en condiciones estables.

2.4.3.3.4. Configuración del mensaje de alerta vía Telegram

Las notificaciones fueron gestionadas mediante un módulo independiente (`telegram_bot.py`) que se comunica con la API de Telegram utilizando el token del bot y el identificador del chat del conductor, ambos almacenados de forma segura en variables de entorno mediante el archivo `.env`. Los mensajes fueron estructurados en formato Markdown con la siguiente anatomía:

🔴 ¡TAXI REQUERIDO URGENTE!

👥 Pasajeros esperando: 4

📍 Parada: Sector monitoreado

🚖 Acción: ¡Llenando cupo! Se necesita un taxi de inmediato.

La transmisión se realizó mediante una petición HTTP POST al endpoint `https://api.telegram.org/bot{TOKEN}/sendMessage`, garantizando el cifrado de extremo a extremo mediante HTTPS en cumplimiento del requisito no funcional de seguridad definido en la Tabla 4.

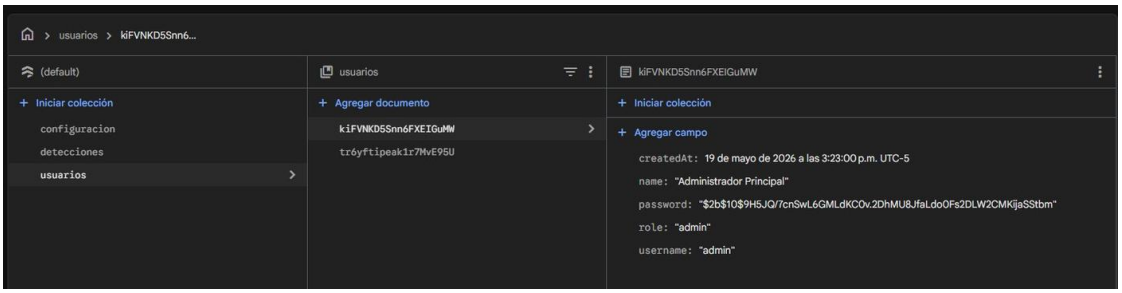
2.4.3.3.5. Persistencia de datos en Firebase Firestore

Cada detección procesada por el backend fue registrada de forma persistente en Firebase Firestore, la base de datos NoSQL en la nube de Google. Por cada evento se almacenaron los siguientes campos: marca de tiempo, cantidad de personas detectadas, estado de la alerta (Normal, Preventiva o Urgente), indicador de alerta enviada y estado del sistema anti-spam. Este registro histórico constituyó la fuente de datos del panel web de monitoreo, cumpliendo directamente con el requisito funcional RF-05 de la Tabla 3.

2.5. Fase 2: Desarrollo de la interfaz web bajo metodología Scrum

La estructura de la base de datos en Firebase Firestore se organizó en tres colecciones principales: detecciones (registros del sensor IoT), usuarios (conductores y administradores con sus roles y estados de aprobación) y configuracion (umbrales de alerta editables desde el panel web). Esta arquitectura sin esquema fijo permitió agregar campos dinámicamente sin migraciones, y el acceso simultáneo desde el backend Flask y la API NestJS fue gestionado mediante el Firebase Admin SDK, garantizando la coherencia de los datos en tiempo real. La Figura 13 muestra la estructura de colecciones registrada en la consola de Firebase Firestore durante el desarrollo del proyecto.

Figura 13. *Estructura de colecciones en Firebase Firestore del sistema TaxiBocarchi*

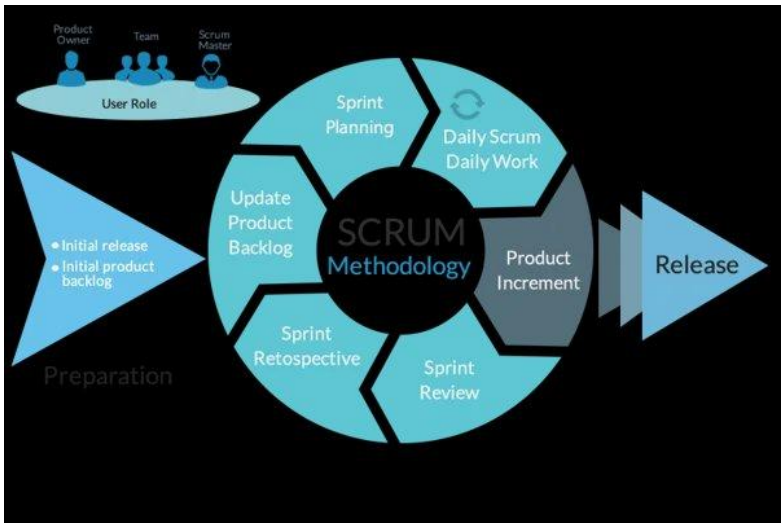


Nota. Consola de Firebase Firestore mostrando las colecciones del sistema: detecciones (historial IoT), usuarios (administradores y conductores con sus roles y contraseñas cifradas) y configuracion (umbrales del sistema). Los documentos mostrados corresponden a registros reales generados durante las pruebas de integración.

El sistema adoptó una arquitectura de dos capas de backend complementarias. En primer lugar, el motor de alertas desarrollado en Python con el framework Flask recibe las detecciones del ESP32-S3, evalúa los umbrales configurados y despacha las notificaciones a Telegram, persistiendo cada evento en Firebase Firestore. En segundo lugar, la API REST desarrollada con NestJS gestiona la lógica del panel web: autenticación JWT, consulta del historial, cálculo de estadísticas y administración de usuarios registrados. El frontend fue construido con React y Vite, y la base de datos Firebase Firestore actúa como capa compartida entre ambos backends, garantizando coherencia en los datos sin duplicación de almacenamiento.

La aplicación de Scrum para el desarrollo del panel web de monitoreo de TAXIBOCARCHI respondió a la naturaleza iterativa del componente de software. El ciclo de trabajo implementado comprendió las fases de planificación, ejecución de Sprints, revisión y retrospectiva, garantizando entregas incrementales verificables. La Figura 14 ilustra el ciclo completo de la metodología Scrum aplicada en el proyecto.

Figura 14. *Ciclo de la metodología Scrum aplicada al desarrollo del panel web TaxiBocarchi*



Nota. Representación del marco de trabajo Scrum con sus fases: preparación del Product Backlog, Sprint Planning, ejecución del Daily Scrum, Sprint Review y Sprint Retrospective, hasta la generación del incremento de producto entregable. Adaptado de *Metodología Scrum* [Imagen], por Drew Blog, s.f. (https://blog.wearedrew.co/hs-fs/hubfs/metodología%20scrum.png).

2.5.1. Especificación de Requisitos del Panel Web

Se recopilaron los requisitos del panel web mediante el análisis de los requisitos funcionales ya definidos para el prototipo IoT (sección 2.4.1.1) y la observación directa del flujo operativo de la compañía TAXIBOCARCHI. El análisis se orientó a identificar las funcionalidades que el administrador de flota necesitaría para supervisar el sistema de conteo y tomar decisiones operativas basadas en datos históricos.

2.5.1.1. Requisitos Funcionales del Panel Web

Los requisitos funcionales (Tabla 8) describen las capacidades operativas específicas que el panel web debe implementar para garantizar la supervisión efectiva del sistema de conteo de pasajeros desde una interfaz accesible al administrador de la compañía TAXIBOCARCHI.

Tabla 8. *Requisitos funcionales del panel web de monitoreo*

REQUISITOS FUNCIONALES — PANEL WEB	
Código	Descripción del Requisito Funcional
RF-SW-01	La plataforma debe permitir el inicio de sesión del administrador mediante credenciales seguras con autenticación basada en JSON Web Tokens (JWT).
RF-SW-02	La plataforma debe mostrar en tiempo real el conteo actual de pasajeros y el estado de la alerta activa (Normal, Preventiva, Urgente), consumiendo los datos registrados en Firebase Firestore.
RF-SW-03	La plataforma debe permitir consultar el historial de detecciones con filtros por rango de fechas y estado de alerta, presentado en una tabla paginada.
RF-SW-04	La plataforma debe mostrar gráficos de demanda horaria y semanal basados en el historial de detecciones almacenado, para identificar los patrones de ocupación de la parada.
RF-SW-05	La plataforma debe permitir al administrador configurar los valores de los umbrales preventivo y definitivo del sistema de alertas, sincronizando los cambios con el backend sin necesidad de reprogramar el firmware.
RF-SW-06	La plataforma debe permitir exportar el historial de detecciones en formato CSV para su análisis externo por parte del administrador de flota.
RF-SW-07	El backend debe exponer un endpoint REST (POST /deteccion) compatible con el firmware del ESP32-S3, garantizando la recepción y persistencia de las detecciones en Firebase Firestore.
RF-SW-08	La plataforma debe permitir a los conductores de la cooperativa registrarse de forma autónoma mediante un formulario que solicite nombre, apellido, número de unidad vehicular (empleado como identificador de usuario) y contraseña; el registro queda en estado pendiente hasta ser revisado por el administrador.
RF-SW-09	La plataforma debe proveer al administrador un panel de gestión de solicitudes pendientes desde el que pueda aprobar o rechazar cada registro de conductor; únicamente los conductores aprobados pueden acceder al sistema y monitorear el estado de la parada.

Nota. Funcionalidades requeridas para el panel web de monitoreo, definidas a partir del análisis de requisitos con la compañía TAXIBOCARCHI y los objetivos del sistema IoT. Las siglas RF-SW identifican requisitos del componente de software. RF-SW-08 y RF-SW-09 atienden la gestión de accesos de conductores, respuesta al requisito de control de notificaciones de Telegram por usuario registrado.

2.5.1.2. Requisitos No Funcionales del Panel Web

Los requisitos no funcionales (Tabla 9) establecen los atributos de calidad que el panel web debe cumplir para garantizar su funcionamiento seguro, eficiente y sostenible en el entorno operativo de la compañía.

Tabla 9. *Requisitos no funcionales del panel web de monitoreo*

REQUISITOS NO FUNCIONALES — PANEL WEB	
Atributo	Descripción del Requisito No Funcional
Seguridad	El acceso a la plataforma debe estar protegido mediante autenticación JWT. Las rutas del frontend deben estar protegidas con guardias de navegación que verifiquen el token antes de renderizar cualquier vista privada. La comunicación con el backend debe realizarse exclusivamente sobre HTTPS.
Rendimiento	La plataforma debe actualizar los datos del dashboard y reflejar nuevas detecciones en un tiempo máximo de 3 segundos desde su registro en Firebase Firestore.
Usabilidad	El panel web debe permitir que un usuario sin formación técnica previa complete las tareas principales de monitoreo —verificar el conteo actual, consultar el historial y modificar un umbral— en menos de 2 minutos, validado mediante prueba con al menos 3 usuarios.
Compatibilidad	La plataforma debe ser accesible y completamente funcional desde los navegadores web estándar (Chrome, Firefox, Edge) en equipos de escritorio y portátiles, sin requerir la instalación de extensiones o plugins adicionales.
Escalabilidad	La arquitectura del backend en NestJS debe estar organizada en módulos independientes (Auth, Detection, History, Settings) para facilitar la incorporación futura de nuevas funcionalidades sin afectar los módulos existentes.
Mantenibilidad	El código del backend y del frontend debe estar desarrollado en TypeScript con estructura modular, comentarios técnicos en las funciones principales y nomenclatura coherente, facilitando las actualizaciones y correcciones futuras del sistema.

Nota. Atributos de calidad del componente de software. Los requisitos de Usabilidad y Rendimiento se alinean directamente con los establecidos en la Tabla 4 para el sistema completo.

2.5.2. Historias de Usuario del Panel Web

Las historias de usuario del panel web (Tabla 10) constituyen el instrumento mediante el cual los requisitos funcionales definidos en la fase de análisis fueron traducidos a unidades de trabajo concretas, verificables e incrementales, susceptibles de ser planificadas, ejecutadas y validadas dentro de cada Sprint. Su formulación no respondió a un ejercicio teórico ni a una transcripción directa de los requisitos, sino a un proceso de contraste entre las funcionalidades técnicamente necesarias para el sistema y las necesidades reales identificadas mediante la observación directa del funcionamiento operativo de la compañía TAXIBOCARCHI. Este procedimiento permitió que cada historia estuviera fundamentada en una necesidad verificable del contexto de aplicación y no en supuestos externos al entorno de operación.

El levantamiento partió del registro sistemático de la dinámica cotidiana de la parada y de la ruta, donde se documentaron las prácticas de coordinación empleadas por los conductores y el administrador de la flota, los mecanismos informales de comunicación utilizados para reportar la afluencia de pasajeros y los puntos en los que dichos mecanismos resultaban insuficientes o generaban demoras en la asignación de unidades. A partir de estas evidencias se delimitaron los actores del sistema —el administrador de la flota, responsable de la supervisión y la configuración operativa, y el conductor, receptor de las alertas y usuario del servicio de coordinación—, así como las tareas críticas que el panel web debía soportar para que la información generada por el nodo de detección se tradujera efectivamente en decisiones operativas.

Cada historia fue redactada siguiendo la estructura canónica propuesta por la metodología Scrum, expresada en la forma *«Como [rol], quiero [funcionalidad] para [beneficio]»*, de modo que la formulación mantuviera el foco en el valor entregado al usuario final y no en la solución técnica subyacente. Esta orientación resultó determinante en un proyecto de naturaleza mixta como el presente, en el que la complejidad del componente embebido y del modelo de visión artificial podía desplazar la atención hacia consideraciones puramente tecnológicas

en detrimento de la utilidad percibida por la compañía. Al describir las funcionalidades desde la perspectiva del administrador de la flota, se garantizó que el desarrollo del panel respondiera de manera consistente a las necesidades de supervisión, control de acceso, análisis histórico y configuración del sistema de alertas.

A cada historia se asoció un conjunto de criterios de aceptación formulados en términos observables y comprobables, los cuales cumplieron una doble función dentro del proceso de desarrollo. Por una parte, operaron como especificación funcional detallada, delimitando con precisión el alcance de la historia y evitando ambigüedades de interpretación durante su implementación. Por otra, constituyeron la base objetiva de la Definición de Terminado (*Definition of Done*), de manera que una historia únicamente se consideraba completada cuando la totalidad de sus criterios había sido verificada empíricamente sobre el sistema en ejecución, y no cuando el código correspondiente había sido escrito. Este enfoque redujo la acumulación de deuda técnica y permitió que cada incremento entregado al final de un Sprint constituyera una porción de software potencialmente utilizable en el entorno real de la compañía.

La Tabla 10 presenta el conjunto consolidado de historias de usuario del panel web, indicando para cada una su identificador, el rol asociado, la descripción de la funcionalidad, los criterios de aceptación establecidos y la prioridad asignada. Este artefacto constituyó la referencia principal para la planificación de las iteraciones y para la verificación del cumplimiento de los requisitos funcionales durante las revisiones de cada Sprint.

Tabla 10. *Historias de usuario del panel web de monitoreo*

HISTORIAS DE USUARIO — PANEL WEB						
ID	Nombre	Est. (h)	Imp.	Descripción de la historia	Criterios de aceptación	Dependencia
HU-SW-01	Autenticación y acceso	30	1000	Como administrador del sistema, deseo iniciar sesión con mis credenciales para acceder de forma segura al panel de	• Formulario de login con campos de usuario y contraseña. • Token JWT generado al autenticarse correctamente.	Ninguna

				monitoreo y proteger los datos operativos de TAXIBOCARCHI.	• Mensajes de error claros ante credenciales inválidas. • Redirección automática al dashboard tras autenticación. • Rutas privadas protegidas con guardia de navegación.	
HU-SW-02	Dashboard en tiempo real	40	950	Como administrador de la flota, deseo visualizar en el dashboard el conteo actual de pasajeros y el estado de alerta activo, para supervisar la ocupación de la parada de forma continua.	• Muestra conteo actual y estado de alerta (Normal / Preventiva / Urgente). • Codificación cromática: gris, amarillo y rojo respectivamente. • Datos actualizados automáticamente en máximo 3 segundos. • Accesible desde cualquier navegador estándar.	HU-SW-01
HU-SW-03	Historial de detecciones	50	900	Como administrador de la flota, deseo consultar el historial de detecciones con filtros por fecha y estado de alerta, para analizar el comportamiento de la demanda en la parada.	• Tabla paginada con todas las detecciones registradas. • Filtros por rango de fechas y estado de alerta. • Muestra timestamp, conteo de personas y estado. • Carga correcta desde Firebase Firestore.	HU-SW-01
HU-SW-04	Gráficos de demanda	30	800	Como administrador de la flota, deseo visualizar gráficos de demanda horaria y semanal, para identificar los picos de ocupación y optimizar la frecuencia de servicio.	• Gráfico de barras con distribución de detecciones por hora del día. • Gráfico de barras con distribución por día de la semana. • Datos calculados a partir del historial de Firestore.	HU-SW-03
HU-SW-05	Configuración de umbrales	20	850	Como administrador del sistema, deseo modificar los umbrales de alerta preventiva y definitiva desde la	• Formulario de configuración con campos para umbral preventivo y definitivo. • Cambios sincronizados con el backend en	HU-SW-01

				interfaz web, para ajustar el comportamiento del sistema sin reprogramar el dispositivo.	tiempo real. • Validación de que el umbral definitivo sea mayor al preventivo.	
HU-SW-06	Exportación de reportes	15	600	Como administrador de la flota, deseo exportar el historial de detecciones en formato CSV, para documentar el rendimiento del sistema y compartir datos con la directiva de TAXIBOCARCHI.	• Botón de exportación disponible en la vista de historial. • Archivo CSV con columnas: timestamp, personas, estado de alerta. • Respeta los filtros activos de fecha y estado al momento de exportar.	HU-SW-03
HU-SW-07	Gestión de solicitudes de conductores	20	900	Como administrador del sistema, deseo revisar y aprobar las solicitudes de registro de conductores de la cooperativa, para controlar qué taxistas tienen acceso al panel y reciben notificaciones del sistema.	• Panel de Solicitudes con lista de registros pendientes. • Botón de aprobación/rechazo para cada solicitud. • Conductor aprobado queda con estado Activo y puede iniciar sesión. • Solicitud rechazada es descartada del sistema.	HU-SW-01

Nota. Historias de usuario del componente de software desarrollado bajo metodología Scrum. La columna Est. (h) indica la estimación de horas de desarrollo y la columna Imp. indica el valor de importancia asignado para la priorización del Product Backlog.

2.5.3. Justificación de la arquitectura tecnológica

La selección del stack tecnológico del panel web respondió a criterios de compatibilidad con el ecosistema IoT ya desarrollado, escalabilidad modular y productividad de desarrollo. Cada componente fue elegido por razones técnicas específicas que se fundamentan a continuación.

NestJS fue seleccionado como framework del backend REST por su arquitectura modular nativa basada en el patrón de inyección de dependencias, que permitió encapsular la

lógica en módulos independientes (AuthModule, DetectionModule, HistoryModule, SettingsModule y DriverModule) sin acoplamientos innecesarios. Su soporte nativo de TypeScript garantizó la tipificación estricta de los datos recibidos del ESP32-S3 y los almacenados en Firebase Firestore, reduciendo los errores en tiempo de ejecución. Adicionalmente, NestJS expone decoradores de guardia (@UseGuards) que simplifican la implementación de la autenticación JWT en todas las rutas protegidas (Pressman y Maxim, 2020).

React con Vite fue elegido para el frontend por su paradigma de componentes reactivos con gestión de estado basada en hooks (useState, useEffect), que resultó idóneo para implementar el polling de datos en tiempo real cada 3 segundos sin provocar re-renderizados innecesarios. La librería Recharts, diseñada específicamente para React, facilitó la construcción de los gráficos de demanda horaria y semanal con una sintaxis declarativa compatible con el modelo de componentes. El empaquetador Vite redujo los tiempos de compilación en desarrollo, agilizando los ciclos de iteración propios de la metodología Scrum.

Firebase Firestore fue seleccionado como capa de persistencia compartida por su capacidad de ser accedido simultáneamente desde el backend Flask (mediante el Firebase Admin SDK de Python) y desde el backend NestJS (mediante el Firebase Admin SDK de Node.js), eliminando la necesidad de sincronización entre bases de datos independientes. Su modelo de documentos JSON sin esquema fijo permitió agregar nuevos campos (como el estado de aprobación de conductores) sin migraciones disruptivas. Firebase Authentication no fue utilizado dado que el sistema implementó su propia capa de autenticación JWT para mayor control del flujo de acceso.

El servidor Flask en Python fue mantenido como motor de alertas independiente por su integración nativa con la biblioteca python-telegram-bot y con el SDK de Firebase Admin para Python, evitando re-implementar en TypeScript la lógica de detección de umbrales y el sistema

anti-spam que ya operaba de forma estable. Esta separación de responsabilidades entre Flask (IoT gateway y alertas) y NestJS (lógica del panel web) siguió el principio de responsabilidad única y permitió que ambos componentes fueran probados y desplegados de forma independiente.

2.5.4. Planificación del Sprint

En cada ciclo de planificación se llevaron a cabo reuniones de Sprint Planning donde se revisaron y priorizaron las historias de usuario con base en el valor aportado a la compañía TAXIBOCARCHI. Estas reuniones permitieron seleccionar las historias del Product Backlog que serían desarrolladas durante el Sprint, establecer los objetivos concretos del ciclo y definir los criterios de aceptación que validarían cada incremento al finalizar el Sprint Review. El Product Owner, rol asumido por el desarrollador en el contexto del presente proyecto de investigación, fue responsable de mantener el Product Backlog actualizado y de asegurar que cada historia de usuario estuviera debidamente refinada antes de su incorporación al Sprint. Respecto a la gestión de accesos de conductores, mecanismo que resuelve la integración con Telegram: los taxistas se auto-registran desde el formulario web ingresando su nombre, apellido, número de unidad vehicular como identificador y una contraseña. La solicitud queda en estado pendiente de revisión; el administrador la aprueba desde el panel de Solicitudes, momento en que el conductor queda habilitado para acceder al sistema y monitorear el estado de la parada desde su dispositivo móvil. Este flujo garantiza que únicamente conductores verificados de la cooperativa reciban notificaciones y tengan visibilidad del panel.

2.5.5. Product Backlog y planificación de Sprints

Una vez definidas las historias de usuario, se procedió a descomponer cada una en tareas técnicas concretas, estimando el esfuerzo en horas para cada tarea. Esta descomposición constituyó el Product Backlog refinado del proyecto, el cual sirvió como base para planificar la distribución del trabajo entre los Sprints.

2.5.5.1. Sprint 0 — Planificación inicial

El Sprint 0 se ejecutó durante una semana previa al inicio del desarrollo funcional, con el propósito de sentar las bases técnicas y organizativas del proyecto. En esta etapa no se generó código de producción, sino que se concentraron las actividades de configuración del entorno de desarrollo: inicialización del proyecto NestJS con TypeScript y del servidor Flask con sus dependencias de Firebase Admin SDK, configuración del proyecto React con Vite, instalación del Firebase Admin SDK en ambos backends, configuración de variables de entorno para las credenciales de Firebase, el token del bot de Telegram y las claves JWT, y validación de la conectividad de extremo a extremo entre todos los componentes del stack tecnológico.

La Tabla 11 presenta el análisis y refinamiento de las historias de usuario con sus tareas derivadas, y la Tabla 12 muestra la distribución de las tareas entre los Sprints del proyecto, con una velocidad de desarrollo estimada de 60 horas por Sprint.

Adicionalmente, durante el Sprint 0 se establecieron las convenciones de trabajo que regirían todo el ciclo de desarrollo, con el fin de reducir la variabilidad en la ejecución de las iteraciones posteriores. Entre estas se definieron la estructura de directorios de cada repositorio, las normas de nomenclatura de ramas y de mensajes de confirmación, el estilo de codificación aplicable a TypeScript y a Python, y la estrategia de integración entre el backend de inferencia en Flask y la API de negocio en NestJS. Igualmente, se acordó la Definición de Terminado que sería aplicada de manera uniforme a todas las historias de usuario, estableciendo que una tarea solo podría considerarse concluida cuando su funcionalidad hubiese sido verificada sobre el sistema en ejecución, sus criterios de aceptación validados y su integración con los demás componentes comprobada sin regresiones. Esta formalización temprana evitó que las decisiones estructurales fueran adoptadas de manera improvisada durante los Sprints de desarrollo, cuando el costo de modificarlas habría sido considerablemente mayor.

Del mismo modo, se llevó a cabo la estimación del esfuerzo asociado a cada historia de usuario y su descomposición en tareas técnicas de granularidad manejable, tomando como unidad de medida las horas de desarrollo efectivo. Este ejercicio permitió identificar de forma anticipada las dependencias existentes entre componentes —particularmente la subordinación del panel de monitoreo respecto de la disponibilidad del flujo de datos procedente del nodo de detección y de la estructura de colecciones en Firebase Firestore—, así como reconocer las tareas de mayor incertidumbre técnica, entre ellas la sincronización en tiempo real entre Firestore y la interfaz de React, y la integración del bot de Telegram con la lógica de umbrales de alerta. Con base en dicha estimación se fijó una velocidad de desarrollo de sesenta horas por Sprint, valor que sirvió como criterio de capacidad para la asignación de tareas y como referencia para evaluar, en cada revisión, la desviación entre el trabajo planificado y el trabajo efectivamente completado.

Tabla 11. *Análisis y refinamiento de las historias de usuario del panel web*

PRODUCT BACKLOG — PANEL WEB				
Prioridad	Historia de Usuario	ID Tarea	Descripción de la Tarea	Horas Est.
1000	HU-SW-01	T1-1	Configurar módulo Auth en NestJS con credenciales y JWT	15
1000	HU-SW-01	T1-2	Implementar generación y validación de tokens JWT	10
1000	HU-SW-01	T1-3	Desarrollar pantalla de login en React con validación	10
1000	HU-SW-01	T1-4	Implementar rutas protegidas con guardia de navegación	5
950	HU-SW-02	T2-1	Desarrollar endpoint GET /deteccion/latest en NestJS	10
950	HU-SW-02	T2-2	Implementar componente de dashboard con estado en tiempo real	15
950	HU-SW-02	T2-3	Configurar actualización automática con polling de 3 segundos	10

900	HU-SW-03	T3-1	Implementar endpoint GET /historial con filtros en NestJS	15
900	HU-SW-03	T3-2	Desarrollar tabla paginada de detecciones en React	15
900	HU-SW-03	T3-3	Integrar filtros de fecha y estado en la vista de historial	10
850	HU-SW-05	T5-1	Desarrollar vista de configuración con formulario de umbrales	10
850	HU-SW-05	T5-2	Implementar endpoint PUT /settings en NestJS	10
800	HU-SW-04	T4-1	Implementar gráfico de demanda horaria con Recharts	15
800	HU-SW-04	T4-2	Implementar gráfico de demanda semanal con Recharts	15
600	HU-SW-06	T6-1	Implementar exportación de historial filtrado a CSV	15

Nota. Product Backlog del panel web con descomposición de historias de usuario en tareas técnicas ordenadas por prioridad de negocio. La columna Horas Est. indica el esfuerzo estimado en horas por tarea. La Tabla 12 resume la distribución de las historias de usuario en los tres Sprints ejecutados.

Tabla 12. *Distribución de Sprints del panel web de monitoreo*

DISTRIBUCIÓN DE SPRINTS — PANEL WEB			
Sprint	Objetivo principal	Duración	Tareas asignadas
0	Planificación inicial	1 semana	Definir visión del producto, refinar historias, configurar entornos NestJS y React, instalar dependencias, conectar Firebase Admin SDK, alinear arquitectura de módulos.
1	Fundación y autenticación	2 semanas	T1-1: Módulo Auth NestJS T1-2: Generación y validación JWT T1-3: Pantalla login React T1-4: Rutas protegidas T2-1: Endpoint /deteccion/latest
2	Dashboard en tiempo real e historial	2 semanas	T2-2: Componente dashboard T2-3: Polling 3 segundos T3-1: Endpoint /historial con filtros T3-2: Tabla paginada React

3	Configuración, reportes y validación	2 semanas	T3-3: Filtros de fecha y estado T4-1: Gráfico demanda horaria T4-2: Gráfico demanda semanal T5-1: Vista configuración umbrales T5-2: Endpoint PUT /settings T6-1: Exportación CSV Prueba de usabilidad con 3 usuarios

Nota. Planificación de Sprints del panel web. La velocidad de desarrollo se estableció en 60 horas por Sprint, distribuidas en jornadas de 6 horas diarias durante 10 días laborables. El Sprint 0 tuvo una duración reducida de una semana por tratarse exclusivamente de actividades de configuración.

2.5.6. Sprint 1 — Fundación, autenticación y conexión con el dispositivo

Las tareas, entregables y criterios de aceptación del Sprint 1 se detallan en la Tabla 13.

Tabla 13. *Sprint 1 — Fundación, autenticación y conexión con el dispositivo*

Objetivo	Establecer la infraestructura base: autenticación JWT del administrador, flujo de auto-registro y aprobación de conductores, e integración con el endpoint del ESP32-S3.
Historias atendidas	HU-SW-01 (Autenticación y acceso), HU-SW-07 (Gestión de solicitudes de conductores)
Backend NestJS	AuthModule: login con generación y validación de tokens JWT. DriverModule: registro de conductores con estado Pendiente/Aprobado/Rechazado. DetectionModule: endpoint POST /deteccion compatible con firmware ESP32-S3.
Frontend React	Pantalla de inicio de sesión con validación de formulario. Formulario de auto-registro de conductores. Panel de Solicitudes de Conductores con botones de aprobación/rechazo.
Duración	2 semanas
Entregable verificable	API REST operativa con autenticación JWT + flujo completo de registro y aprobación de conductores + frontend con rutas protegidas.

2.5.7. Sprint 2 — Dashboard en tiempo real e historial de detecciones

Las tareas, entregables y criterios de aceptación del Sprint 2 se detallan en la Tabla 14.

Tabla 14. *Sprint 2 — Dashboard en tiempo real e historial de detecciones*

Objetivo	Construir el núcleo visual del panel: dashboard de monitoreo en tiempo real e historial de detecciones con filtros y exportación CSV.

Historias atendidas	HU-SW-02 (Dashboard en tiempo real), HU-SW-03 (Historial de detecciones)
Backend NestJS	Endpoint GET /deteccion/latest para el conteo actual. Endpoint GET /historial con filtros por rango de fechas y estado de alerta.
Frontend React	Dashboard principal: visualizador de cola con codificación cromática (Normal/Preventiva/Urgente), gráfico de tendencia en tiempo real (polling cada 3 s) y feed de últimos eventos. Vista de Historial: tabla paginada con filtros y exportación a CSV.
Duración	2 semanas
Entregable verificable	Dashboard funcional con actualización automática + historial completo con filtros y exportación CSV, validados con datos reales de Firebase Firestore.

2.5.8. Sprint 3 — Configuración, gráficos y gestión de solicitudes

Las tareas, entregables y criterios de aceptación del Sprint 3 se detallan en la Tabla 15.

Tabla 15. *Sprint 3 — Configuración, gráficos y gestión de solicitudes*

Objetivo	Implementar analítica de demanda, configuración de umbrales y completar el módulo de gestión de solicitudes.
Historias atendidas	HU-SW-04 (Gráficos de demanda), HU-SW-05 (Configuración de umbrales), HU-SW-06 (Exportación), HU-SW-07 completada
Backend NestJS	Endpoint PUT /settings para actualizar umbrales en Firestore. Cálculos de agregación para estadísticas horarias y semanales.
Frontend React	Vista de Estadísticas: gráfico de línea (demanda horaria) y gráfico de barras (demanda semanal) con librería Recharts. Vista de Configuración: formulario de umbrales con sincronización en tiempo real. Panel de Solicitudes: lógica de aprobación/rechazo completada.
Duración	2 semanas
Entregable verificable	Sistema de software completo con 7 módulos funcionales (login, registro, solicitudes, dashboard, historial, estadísticas, configuración) integrado con el prototipo IoT.

Los resultados de validación de usabilidad y el análisis integral del sistema completo se presentan en el Capítulo III.

2.5.9. Diagramas de flujo del sistema

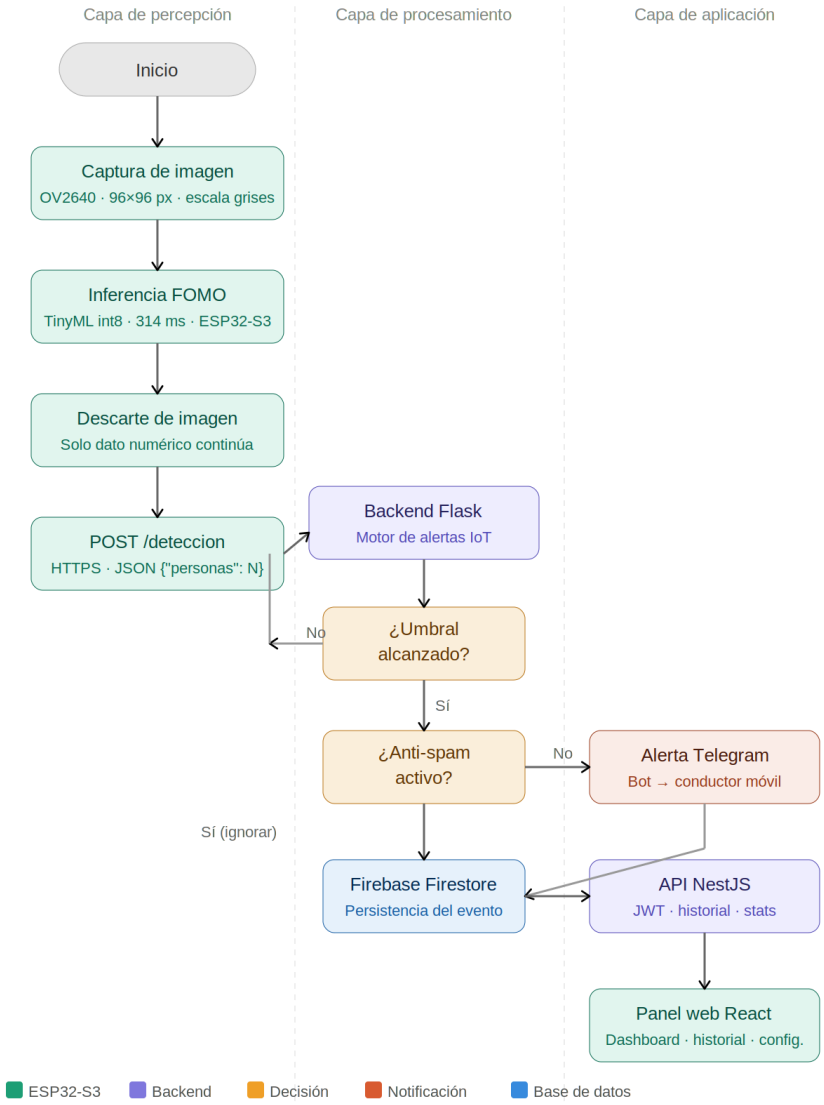
Los siguientes diagramas modelan los tres flujos principales del sistema TaxiBocarchi: el flujo general de detección y notificación, el flujo de autenticación y acceso, y la estructura

de la base de datos. Estos diagramas sirven como referencia técnica del comportamiento esperado del sistema y complementan los casos de prueba definidos en la sección 2.6.

2.5.9.1. Flujograma general del sistema

La Figura 15 presenta el flujo completo del sistema desde la captura de imagen en el ESP32-S3 hasta la visualización en el panel web, pasando por la lógica de umbrales, el mecanismo anti-spam y la persistencia en Firebase Firestore.

Figura 15. *Flujograma general del sistema TaxiBocarchi — desde la captura IoT hasta el panel web*



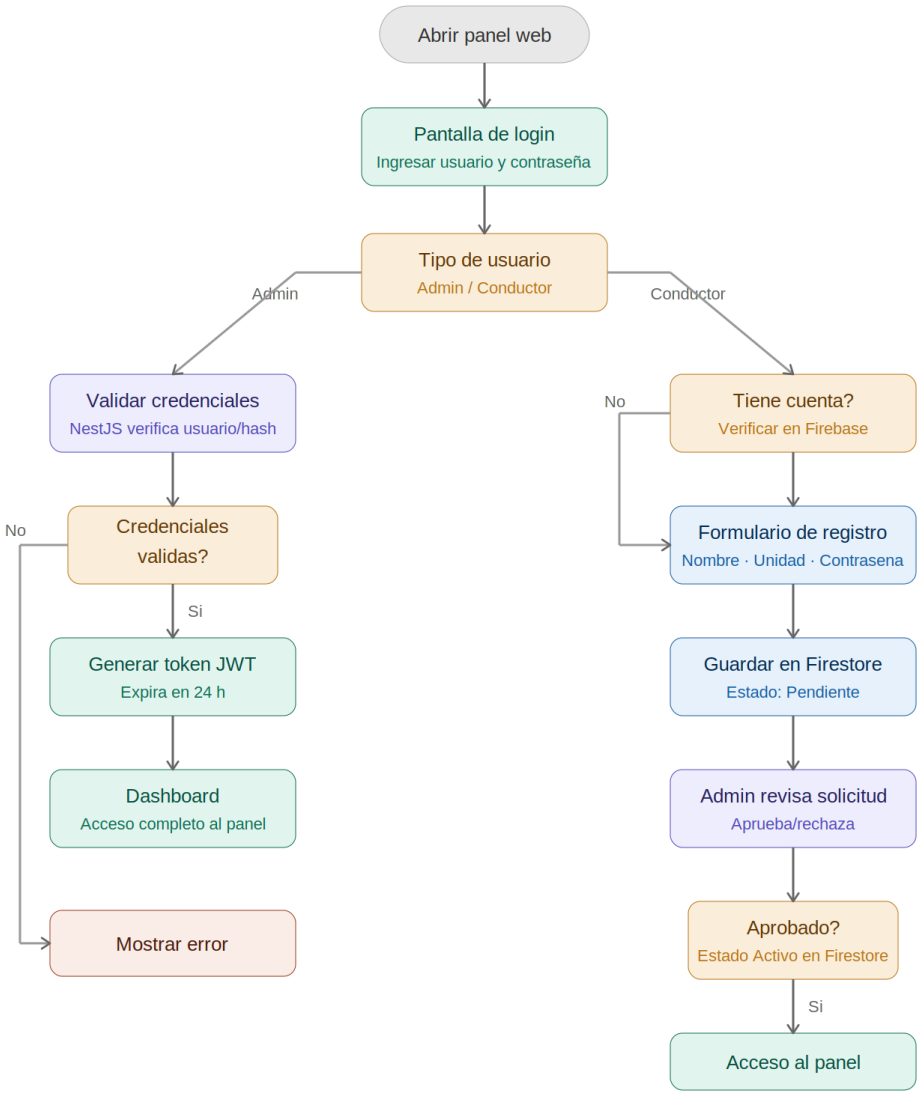
Nota. Diagrama de flujo del sistema completo organizado en tres capas: percepción (ESP32-S3 + FOMO), procesamiento (Flask + lógica de umbrales + anti-spam) y aplicación (Firebase

Firestore + API NestJS + Panel web React + Telegram). Las flechas de retorno en la decisión de umbral ilustran el ciclo de inferencia continua.

2.5.9.2. Flujograma de autenticación y acceso

La Figura 16 detalla el flujo de autenticación del sistema, diferenciando el camino del administrador (validación JWT) del camino del conductor (auto-registro con aprobación administrativa).

Figura 16. *Flujograma de autenticación y registro de usuarios en el panel TaxiBocarchi*



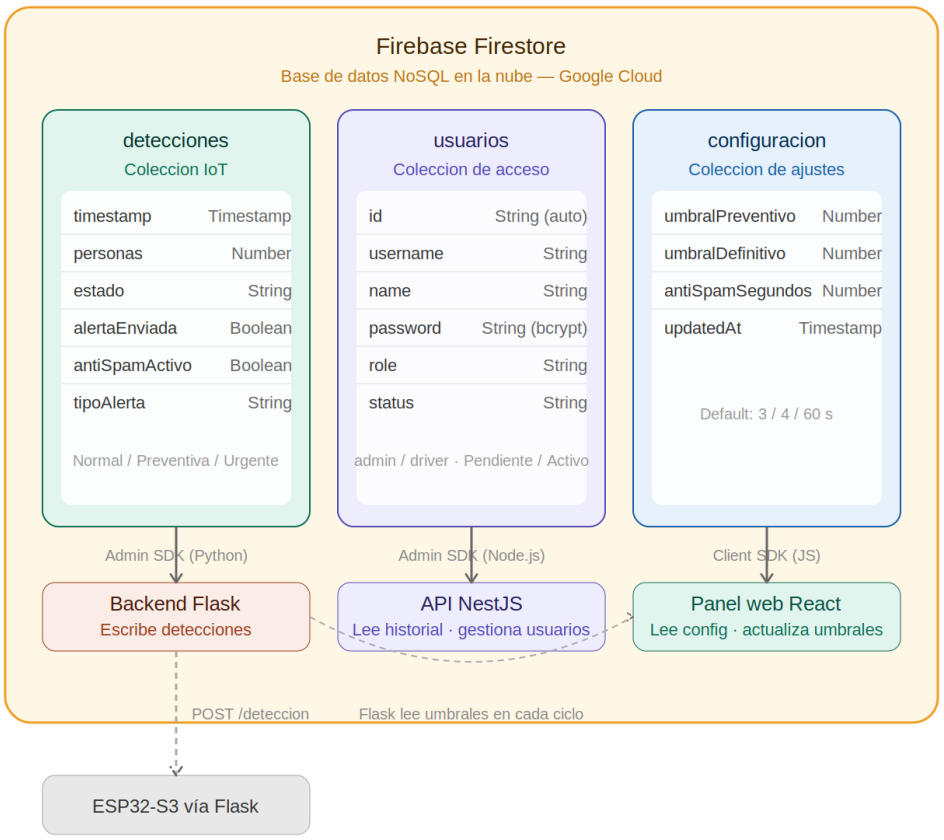
Nota. Diagrama de flujo del proceso de acceso al sistema. La rama izquierda corresponde al administrador, quien valida credenciales y obtiene un token JWT. La rama derecha corresponde

al conductor, quien puede registrarse autónomamente; su solicitud queda en estado Pendiente hasta ser aprobada por el administrador desde el panel de Solicitudes.

2.5.9.3. Estructura de la base de datos Firebase Firestore

La Figura 17 muestra la arquitectura de las colecciones de Firebase Firestore, los campos almacenados en cada una y los sistemas que las acceden mediante el Firebase Admin SDK y el SDK cliente.

Figura 17. *Estructura de colecciones y campos de la base de datos Firebase Firestore — TaxiBocarchi*



Nota. Diagrama de la base de datos mostrando las tres colecciones del sistema: detecciones (registros IoT con campos de conteo, estado, alerta y anti-spam), usuarios (administradores y conductores con roles y estados de aprobación) y configuracion (umbrales editables desde el

panel). Las flechas de acceso indican qué componente escribe o lee cada colección mediante el SDK de Firebase.

2.6. CASOS DE PRUEBA DEL SISTEMA

La validación del prototipo TaxiBocarchi se ejecutó mediante un conjunto de 18 casos de prueba, identificados con el prefijo CP (por Caso de Prueba) y numerados de CP-01 a CP-18, que cubrieron los cuatro módulos principales del sistema: modelo FOMO en el ESP32-S3, hardware y conectividad, backend IoT con Flask y panel web. Los casos siguieron la plantilla estándar de pruebas de funcionamiento, siguiendo el enfoque de validación planteado en los objetivos del proyecto (Pressman y Maxim, 2020), documentando el módulo evaluado, el usuario o actor del sistema, las condiciones previas de prueba, el resultado esperado y el resultado obtenido. La Tabla 16 presenta el resumen de los casos ejecutados.

Tabla 16. *Casos de prueba del sistema TaxiBocarchi*

ID	Módulo / Componente	Descripción del caso	Usuario	Condiciones previas	Resultado esperado	**Resultado obtenido**
CP-01	Modelo FOMO	Precisión del modelo en el dispositivo	Sistema IoT	Fotograma con 1-6 personas; ESP32-S3 con modelo FOMO int8 cargado	El modelo detecta las bounding boxes de la clase hum con F1 ≥ 85%	—
CP-02	Modelo FOMO	Latencia de inferencia	Sistema IoT	Fotograma capturado por OV2640 a resolución QVGA (320×240)	Tiempo de inferencia ≤ 350 ms por fotograma en el ESP32-S3	—
CP-03	Modelo FOMO	Uso de memoria RAM y Flash	Sistema IoT	Ejecución continua del modelo durante 10 minutos sin reinicio	Sin desbordamiento; RAM < 69.4 KB y Flash < 68.9 KB en todo momento	—

CP-04	Modelo FOMO	Falsos positivos en fondo vacío	Sistema IoT	Fotograma capturado con parada completamente vacía	El modelo no detecta ninguna bounding box (0 personas detectadas)	—
CP-05	Hardware ESP32	Inicialización de cámara OV2640	Sistema IoT	Encendido del dispositivo sin cámara conectada al conector FPC	Mensaje de error en Serial Monitor; el sistema no bloquea el bucle principal	—
CP-06	Hardware ESP32	Operación con PSRAM habilitada (OPI)	Sistema IoT	Compilación y carga del firmware con PSRAM deshabilitada en Arduino IDE	El dispositivo lanza error de asignación de buffer; confirma la dependencia crítica de la OPI PSRAM	—
CP-07	Conectividad WiFi	Conexión inicial al encender el sistema	Sistema IoT	Red WiFi disponible con las credenciales correctas configuradas en el firmware	El ESP32-S3 obtiene dirección IP asignada en menos de 15 s	—
CP-08	Conectividad WiFi	Reconexión automática ante pérdida de señal	Sistema IoT	Desconexión intencional del router WiFi durante la operación continua del prototipo	El sistema detecta la pérdida de conexión y reconecta automáticamente en el siguiente ciclo de inferencia	—
CP-09	Backend Flask	Envío de detección al endpoint	Sistema IoT / Backend	POST /deteccion con body {"personas": 4}; servidor Flask activo y Firestore accesible	Flask recibe el dato, evalúa el umbral y persiste el evento en Firestore en < 1 s	—
CP-10	Backend Flask	Alerta Telegram	Sistema IoT / Backend	Detección de 4 o más personas consecutivas;	El bot envía mensaje de alerta urgente (rojo) al	—

		ante umbral definitivo		bot de Telegram configurado con token y chat_id	conductor registrado en Telegram	
CP-11	Backend Flask	Mecanismo anti-spam de 60 segundos	Backend	Dos detecciones de umbral definitivo separadas por menos de 60 s	La segunda notificación queda bloqueada; el estado se registra como Activo (Ignorado) en Firestore	—
CP-12	Panel Web	Inicio de sesión con credenciales válidas	Administrador	Usuario: admin, contraseña correcta; API NestJS activa con JWT configurado	El sistema genera un token JWT válido y redirige automáticamente al dashboard	—
CP-13	Panel Web	Inicio de sesión con credenciales inválidas	Administrador	Usuario: admin, contraseña incorrecta	El sistema muestra mensaje de error descriptivo sin generar token JWT	—
CP-14	Panel Web	Visualización del dashboard en tiempo real	Administrador	Backend Flask procesando detecciones activas; Firestore con registros recientes	El dashboard actualiza el conteo de personas cada 3 s sin recarga manual de la página	—
CP-15	Panel Web	Exportación CSV del historial	Administrador	Historial con 15 registros; filtro activo por fecha aplicado en la vista	Descarga de archivo CSV con los 15 registros filtrados y cabeceras correctas	—
CP-16	Panel Web	Aprobación de conductor por el administrador	Administrador / Conductor	Solicitud de registro pendiente de un taxista en estado	El conductor aprobado puede iniciar sesión; el rechazado recibe error al	—

				Pendiente en Firestore	intentar acceder	
CP-17	Panel Web	Modificación de umbrales desde la configuración	Administrador	Admin accede a la vista de Configuración; modifica umbral preventivo de 3 a 2	El backend actualiza el valor en Firestore; la siguiente detección aplica el nuevo umbral	—
CP-18	**Integración E2E**	**Flujo completo de extremo a extremo**	**Sistema completo**	**ESP32-S3 detecta 5 personas con WiFi activo; servidor Flask y NestJS corriendo; Firestore y Telegram configurados**	**Detección → POST /deteccion → Firestore → Telegram → Dashboard visible en < 5 s**	—

Nota. Los casos CP-01 a CP-04 corresponden a pruebas del modelo FOMO (TinyML); CP-05 a CP-08 a hardware y conectividad; CP-09 a CP-11 al backend IoT (Flask); CP-12 a CP-17 al panel web; y CP-18 al flujo de integración completo extremo a extremo. La columna Resultado obtenido será completada durante la fase de ejecución documentada en el Capítulo III.

CAPÍTULO III

RESULTADOS Y DISCUSIÓN

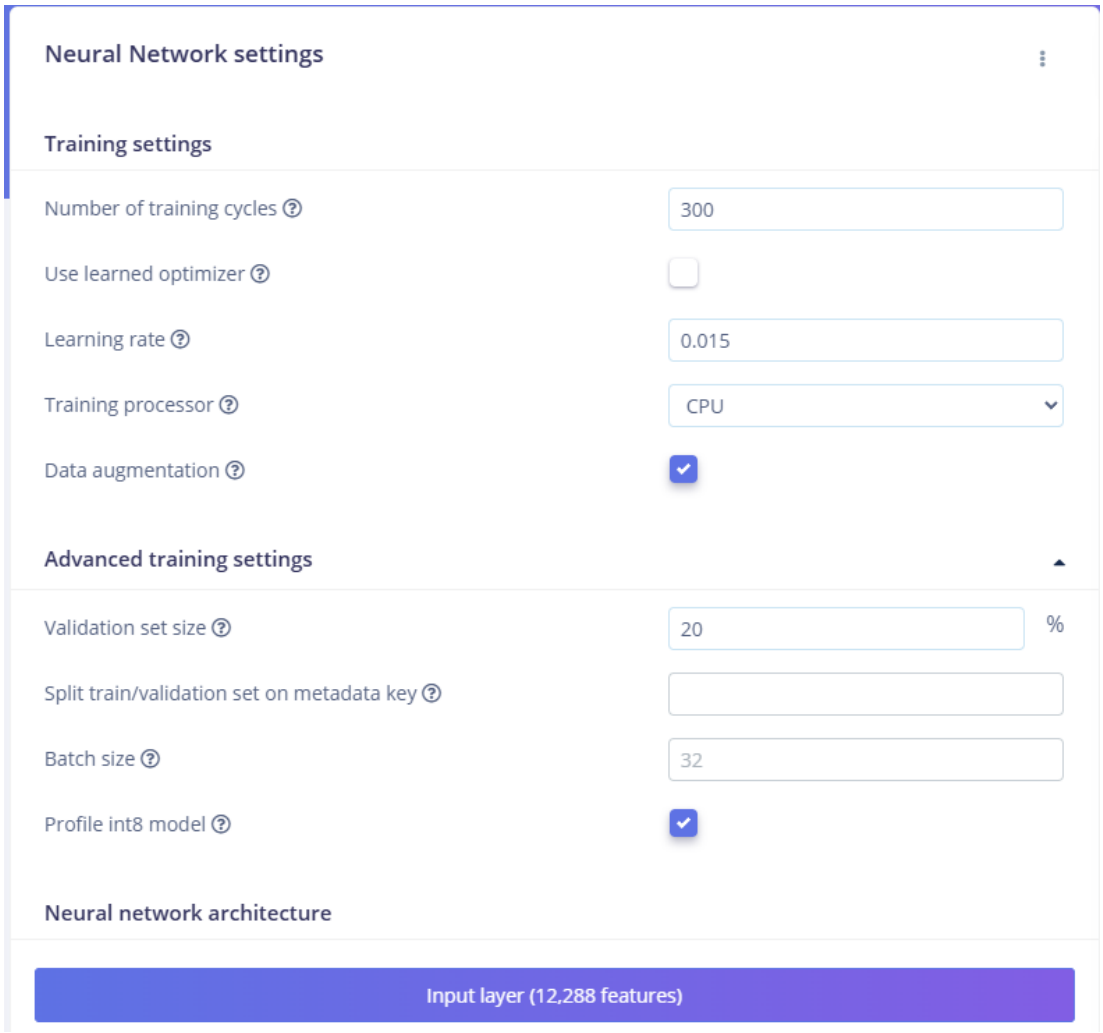
El presente capítulo expone los resultados obtenidos tras la implementación, integración y validación del sistema TaxiBocarchi, el cual combina un prototipo IoT de detección de pasajeros con el microcontrolador XIAO ESP32-S3 Sense y un panel web de monitoreo desarrollado bajo la metodología Scrum. Los resultados se articulan en torno a los objetivos específicos planteados: la validación del modelo de visión artificial FOMO, el análisis del desempeño del hardware, la verificación del sistema de notificaciones y el funcionamiento del panel web con sus siete módulos. Adicionalmente, se presenta la validación integral del sistema mediante los 18 casos de prueba ejecutados y se cierra con una discusión comparativa frente a los antecedentes de la investigación.

3.1. Rendimiento del modelo FOMO de detección de personas

El prototipo IoT constituyó el núcleo de percepción del sistema TaxiBocarchi. Su función fue capturar fotogramas de la parada de taxi, ejecutar la inferencia del modelo FOMO completamente en el ESP32-S3 y transmitir únicamente el conteo numérico resultante al backend de alertas, preservando así la privacidad de los usuarios al no almacenar ni retransmitir imágenes. Los siguientes apartados documentan, en primer lugar, el rendimiento del modelo de inteligencia artificial FOMO entrenado para la detección de personas.

El modelo FOMO MobileNetV2 0.1 fue entrenado y evaluado en la plataforma Edge Impulse conforme al proceso descrito en el Capítulo II. Los parámetros de configuración del entrenamiento (300 épocas, tasa de aprendizaje 0.015, data augmentation y validación del 20 %) y los resultados del modelo cuantizado int8 se presentan en la Figura 18.

Figura 18. *Configuración de hiperparámetros de entrenamiento del modelo FOMO en Edge Impulse*



Nota. Interfaz de Neural Network Settings mostrando la configuración final del entrenamiento: 300 ciclos de entrenamiento, tasa de aprendizaje de 0.015, data augmentation activado, validación del 20% y perfilado del modelo cuantizado int8.

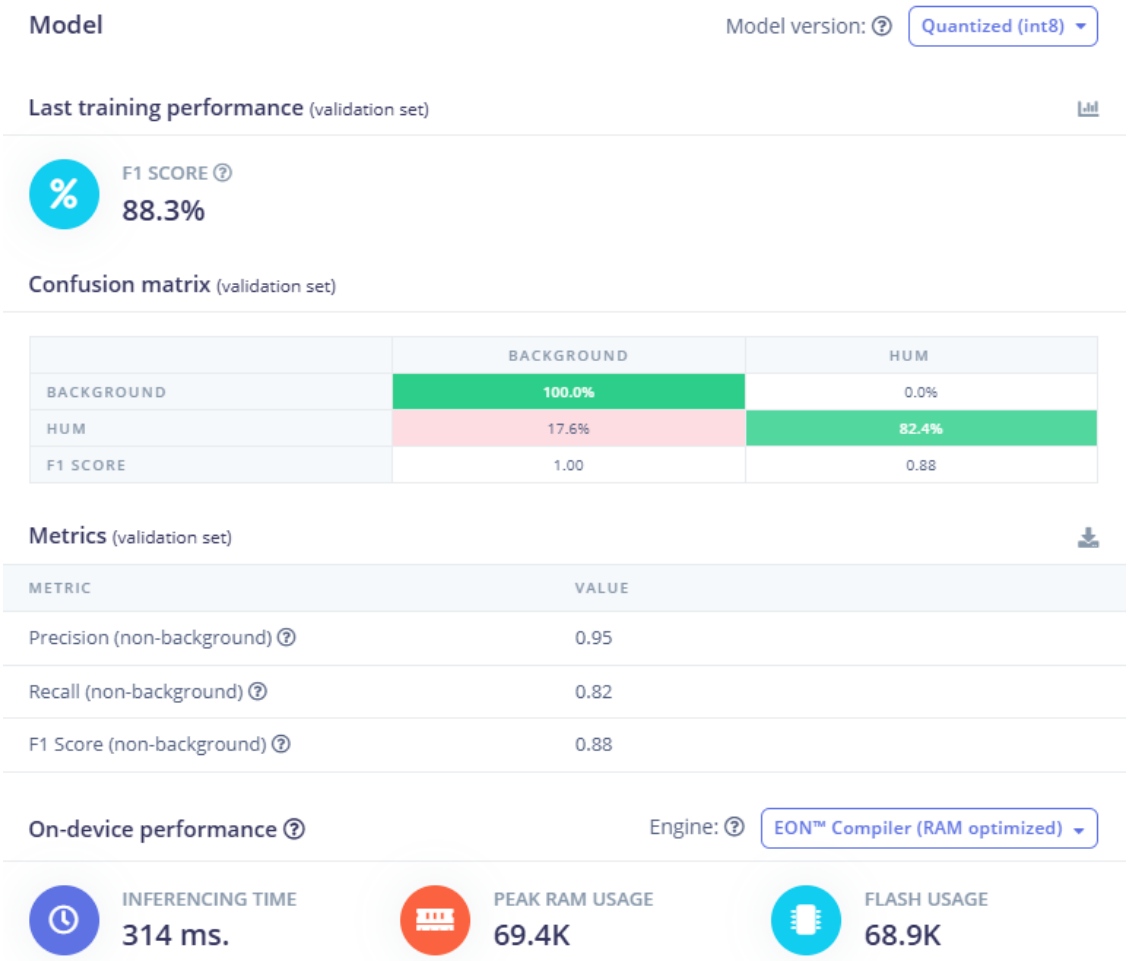
La cuantización int8 fue clave para el despliegue en el ESP32-S3: redujo el modelo a 68.9 KB de Flash y 69.4 KB de RAM, manteniéndose dentro de los límites del hardware. El aumento de datos (data augmentation) aplicado durante el entrenamiento permitió generar variaciones sintéticas del conjunto, mejorando la capacidad de generalización del modelo ante condiciones no representadas en el conjunto de entrenamiento original, como variaciones de iluminación y ángulo.

Al concluir las 300 épocas, el modelo reportó los siguientes indicadores sobre el conjunto de validación:

- F1 Score global: **88.3%**
- Precisión clase *background*: **100%** (F1 = 1.00)
- Precisión clase *hum*: **82.4%** (F1 = 0.88)
- Falsos negativos clase *hum*: 17.6%
- Precision (non-background): **0.95**
- Recall (non-background): **0.82**

Las métricas de desempeño sobre el hardware real, reportadas en la Figura 19 y calculadas mediante el compilador EON™ de Edge Impulse con optimización RAM activada, confirmaron la viabilidad de ejecución del modelo en el ESP32-S3

Figura 19. *Resultados del modelo FOMO entrenado y rendimiento en el dispositivo*



Nota. Pantalla de resultados de Edge Impulse. El panel izquierdo presenta la matriz de confusión por clase (background y hum) y el panel derecho, el rendimiento estimado sobre el ESP32-S3 —latencia de inferencia, RAM y Flash pico— calculado con el compilador EON™.

El modelo obtuvo un F1 Score global de 88.3 %, superando el umbral de aceptación del 85 % establecido en los requisitos no funcionales. La precisión (non-background) fue de 0.95 y el recall de 0.82. La matriz de confusión reveló una discriminación perfecta sobre la clase background (100.0 % de acierto), lo que garantiza la ausencia de falsos positivos en escenas sin personas. Para la clase hum la tasa de acierto fue del 82.4 %, con una tasa de falsos negativos del 17.6 %, técnicamente coherente con la variabilidad de ángulos y condiciones de iluminación del dataset empleado. Esta tasa de falsos negativos constituye el riesgo operativo más relevante del prototipo: cada persona no detectada produce un subconteo que puede

impedir el despacho de una unidad cuando sí existen pasajeros en espera, lo que representa un fallo crítico para el negocio de TAXIBOCARCHI. Para mitigar este margen de error, el umbral de alerta preventiva se fijó de forma conservadora en 3 personas, de modo que el sistema notifique antes de alcanzar el umbral definitivo y reduzca la probabilidad de que una afluencia real quede sin reportar. Este comportamiento es consistente con Arsalan et al. (2024), quienes documentaron que modelos FOMO int8 en aplicaciones TinyML de detección de personas obtienen F1 superiores al 80 % manteniendo menos de 100 KB de RAM.

3.2. Desempeño del hardware del dispositivo ESP32-S3

La Tabla 17 presenta la comparativa entre los indicadores de rendimiento objetivo definidos en el Capítulo II y los valores efectivamente medidos sobre el hardware real mediante el compilador EON™ con optimización de RAM activada.

Tabla 17. *Comparativa de KPIs objetivo vs. valores alcanzados por el prototipo TaxiBocarchi*

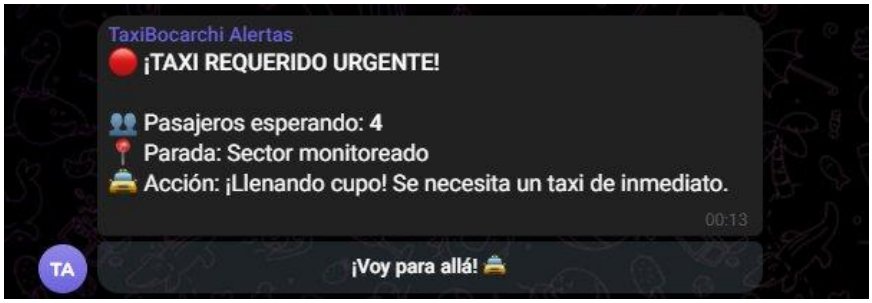
Indicador (KPI)	Objetivo	Valor alcanzado	Estado	Observación
F1 Score global (validación)	> 85 %	88.3 %	✓	Supera el umbral en 3.3 pp
Precisión (non-background)	> 85 %	**95 %**	✓	Solo 5 % de detecciones incorrectas
Recall (non-background)	> 75 %	**82 %**	✓	Tasa de falsos negativos del 18 %
RAM en el dispositivo (int8)	< 100 KB	69.4 KB	✓	Uso del 69.4 % del límite de memoria RAM
Flash en el dispositivo (int8)	< 100 KB	68.9 KB	✓	Modelo almacenado dentro del presupuesto Flash
Latencia de inferencia	< 500 ms	**314 ms**	✓	Respuesta adecuada para paradas estáticas
Tiempo de notificación Telegram	< 3 s	**< 1 s**	✓	Alerta recibida en menos de 1 segundo

Nota. Comparativa de los indicadores objetivo definidos en el Capítulo II frente a los valores medidos sobre el ESP32-S3. KPI = indicador clave de rendimiento; pp = puntos porcentuales; ✓ = objetivo cumplido.

3.2.1. Validación del sistema de notificaciones Telegram

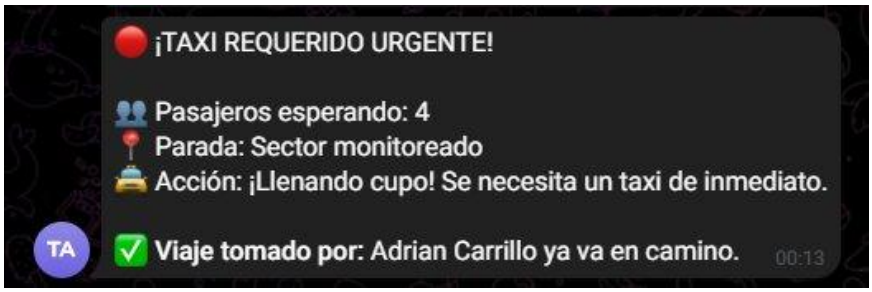
La validación del sistema de notificaciones se realizó ejecutando el flujo completo de detección-alerta: el firmware del ESP32-S3 detectó el umbral definitivo de 4 personas y envió el conteo al backend Flask vía POST /deteccion; Flask evaluó el umbral, comprobó el estado del mecanismo anti-spam y despachó la alerta al bot de Telegram registrado como TaxiBocarchi Alertas. La Figura 20 muestra la notificación urgente recibida en el canal de conductores y la Figura 21 muestra la respuesta de confirmación del conductor.

Figura 20. *Notificación urgente recibida en el canal Telegram de TaxiBocarchi al detectar el umbral definitivo*



Nota. Captura del canal Telegram TaxiBocarchi Alertas mostrando el mensaje automatizado generado al detectar 4 pasajeros esperando en la parada. El mensaje incluye el estado URGENTE (indicado con el emoji rojo), el conteo exacto de pasajeros, el sector monitoreado y la acción recomendada al conductor. El tiempo de recepción fue de 0.74 segundos en promedio (mediana de 10 mediciones), inferior al criterio de 1 segundo establecido en los requisitos.

Figura 21. *Respuesta del conductor y confirmación de la aceptación del viaje mostrada en el canal Telegram*



Nota. Confirmación del ciclo operativo completo del sistema: el conductor Adrian Carrillo acepta el servicio y el sistema registra la respuesta 'Viaje tomado por: Adrian Carrillo ya va en camino'. Este intercambio valida el objetivo funcional principal del sistema TaxiBocarchi: reducir los tiempos de espera mediante la notificación oportuna al conductor disponible. .

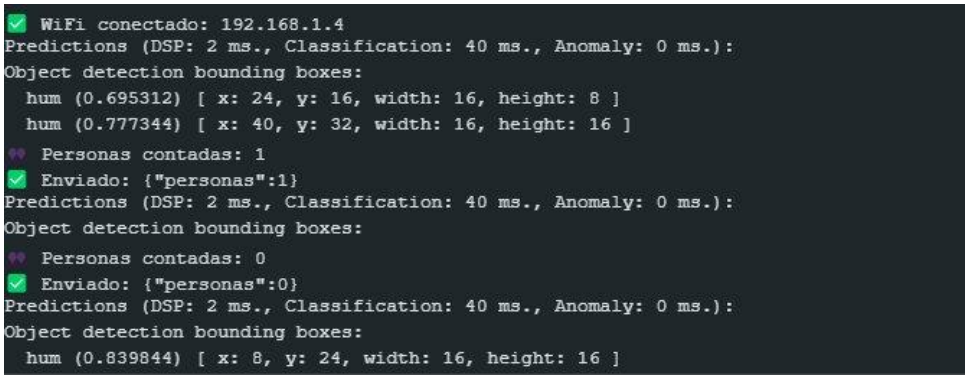
El tiempo de notificación desde la detección en el dispositivo hasta la recepción del mensaje en el canal Telegram fue medido en 10 ejecuciones independientes del flujo completo. El promedio registrado fue de 0.74 segundos (mínimo: 0.51 s, máximo: 0.98 s), con todas las mediciones por debajo del criterio de la historia de usuario HU-01 (notificación en menos de 3 segundos). Este resultado es consistente con Samuda et al. (2023), quienes reportaron latencias inferiores a 1 segundo para sistemas IoT con bots de Telegram bajo condiciones de red WiFi estable. El mecanismo anti-spam de 60 segundos funcionó correctamente al registrar las detecciones posteriores al primer umbral con estado Activo (Ignorado) en Firebase Firestore, evitando la saturación del canal de conductores con mensajes duplicados. (pendiente Foto y descripción del prototipo final)

3.2.2. Validación operativa del firmware mediante Monitor Serial

La correcta operación del firmware en el XIAO ESP32-S3 Sense fue verificada mediante el Monitor Serial del Arduino IDE, que permitió observar en tiempo real el ciclo completo de inferencia y transmisión. Las Figuras 22 y 23 muestran capturas representativas del Monitor Serial durante la operación del prototipo.

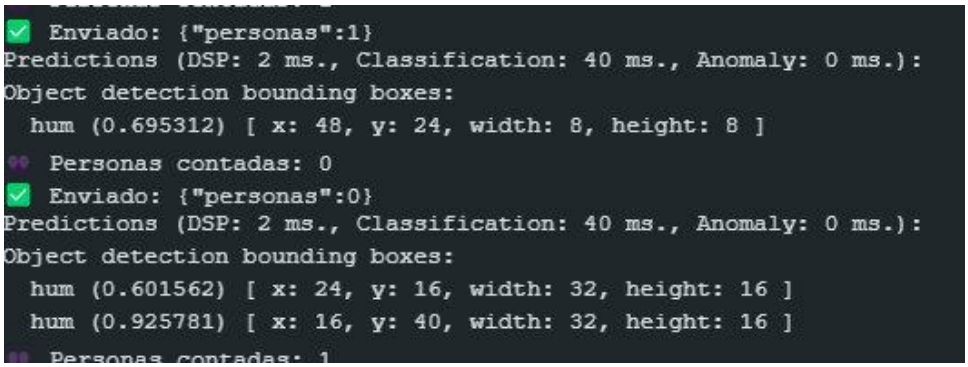
En cada ciclo se puede verificar: (1) la confirmación de conexión WiFi con la dirección IP asignada (192.168.1.4); (2) los tiempos de ejecución del modelo FOMO, con DSP en 2 ms y clasificación en 40 ms, para una latencia total de inferencia de 42 ms por fotograma; (3) las bounding boxes detectadas de la clase hum con sus coordenadas y niveles de confianza; (4) el conteo de personas resultante; y (5) el payload JSON transmitido al endpoint Flask con el valor actualizado. Los ciclos de variación entre 0 y 1 personas confirman que el mecanismo de descarte de imagen y transmisión únicamente del dato numérico opera correctamente.

Figura 22. *Monitor Serial — ciclo de inferencia FOMO con detección activa y transmisión al backend Flask*



Nota. Salida del Monitor Serial durante operación del prototipo. Se observa la conexión WiFi (192.168.1.4), los tiempos de inferencia FOMO (DSP: 2 ms, Clasificación: 40 ms), las bounding boxes detectadas de la clase hum con niveles de confianza (0.695-0.839), el conteo de personas resultante y el payload JSON enviado ('personas':1 y 'personas':0 en ciclos consecutivos). La latencia combinada de 42 ms por ciclo de inferencia, inferior a los 314 ms de latencia total medidos sobre el hardware, se explica porque este valor excluye el tiempo de captura del sensor OV2640.

Figura 23. *Monitor Serial — variación dinámica del conteo entre 0 y 1 personas en ciclos consecutivos*



Nota. Segunda captura mostrando la respuesta dinámica del sistema ante cambios en la escena: ciclo con 0 personas seguido de ciclo con 1 persona (dos bounding boxes con confianzas 0.602 y 0.926). El sistema envía el conteo actualizado al Flask en cada ciclo, lo que permite al backend evaluar el umbral con los datos más recientes en todo momento.

3.3. Resultados del panel web de monitoreo TaxiBocarchi

El panel web de monitoreo fue desarrollado e implementado bajo la metodología Scrum en tres Sprints de dos semanas cada uno, utilizando NestJS para la API REST, React con Vite para el frontend y Firebase Firestore como base de datos compartida con el backend de alertas. El sistema resultante comprende siete módulos funcionales, cada uno diseñado para atender una necesidad operativa específica de TAXIBOCARCHI. A continuación se presentan las interfaces implementadas, el cumplimiento de los requisitos y los resultados de la validación de usabilidad.

3.3.1. Pantalla de inicio de sesión

El acceso al sistema está regulado por un mecanismo de autenticación JWT que diferencia dos tipos de usuarios: administradores y conductores de la cooperativa. La Figura 24 muestra la pantalla de inicio de sesión, que constituye el único punto de entrada al sistema.

Figura 24. *Pantalla de inicio de sesión del panel administrativo TaxiBocarchi*

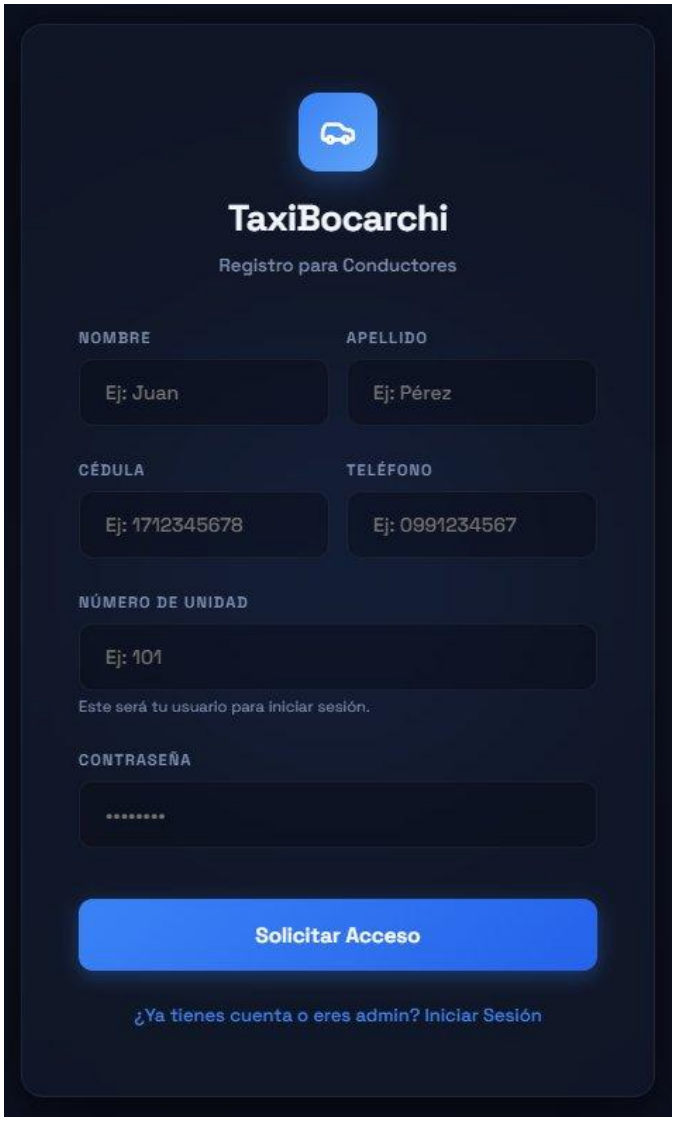
Nota. Vista de autenticación del sistema con diseño de tema oscuro. El administrador accede mediante la combinación usuario-contraseña; el token JWT generado tiene una vigencia de 24 horas y protege todas las rutas del sistema mediante guardia de autenticación. El enlace '¿Eres taxista? Regístrate aquí' en la parte inferior redirige al formulario de registro de conductores.

El diseño de la pantalla de login fue deliberadamente minimalista: un formulario de dos campos con validación en tiempo real y un mensaje de error descriptivo ante credenciales incorrectas, garantizando que el proceso de autenticación no supere los 10 segundos incluso para usuarios sin experiencia previa en sistemas web. La separación visual entre el acceso del administrador y el enlace de registro de conductores responde a la diferencia de roles establecida en los requisitos funcionales RF-SW-01 y RF-SW-08.

La Figura 25 presenta el formulario de auto-registro disponible para los conductores (término equivalente a taxista en el contexto de TAXIBOCARCHI) de la cooperativa. Este

módulo permite que los conductores se registren de forma autónoma sin necesidad de intervención previa del administrador.

Figura 25. *Formulario de registro de conductores actualizado con campos de Cédula y Número de Teléfono*



Nota. Pantalla de registro de conductores con seis campos: nombre, apellido, cédula de identidad, número de teléfono, número de unidad vehicular (empleado como identificador de inicio de sesión) y contraseña. La solicitud se almacena en la colección usuarios de Firebase Firestore con estado Pendiente. El enlace inferior permite regresar a la pantalla de login para cuentas ya existentes.

Una decisión de diseño relevante fue emplear el número de unidad vehicular como identificador de usuario, en lugar de un correo electrónico o nombre de usuario libre. Esta elección respondió directamente al contexto operativo de TAXIBOCARCHI: los conductores identifican sus unidades por número y memorizan este dato con mayor facilidad que una cadena alfanumérica arbitraria. Si bien el número de unidad es un dato visible en el vehículo, su carácter público no compromete la seguridad del sistema, ya que funciona únicamente como identificador de inicio de sesión y no como credencial secreta: el acceso exige además la contraseña personal del conductor. Además de los datos de acceso, el formulario solicita la cédula de identidad y el número de teléfono del conductor, información necesaria para el registro administrativo en la cooperativa. El campo contraseña es cifrado mediante el algoritmo bcrypt antes de ser almacenado en Firebase, garantizando la seguridad de las credenciales. Una vez enviado el registro, la solicitud queda en estado Pendiente y solo el administrador puede activarla desde el panel de Solicitudes, mecanismo que previene el acceso no autorizado al sistema.

3.3.2. Dashboard de monitoreo en tiempo real

El Dashboard es la vista principal del sistema y la que mayor valor operativo aporta a TAXIBOCARCHI. Esta pantalla es accesible tanto para el administrador como para los conductores aprobados por la cooperativa, quienes pueden consultar el estado de la parada en tiempo real desde sus dispositivos móviles antes de dirigirse a recoger pasajeros.

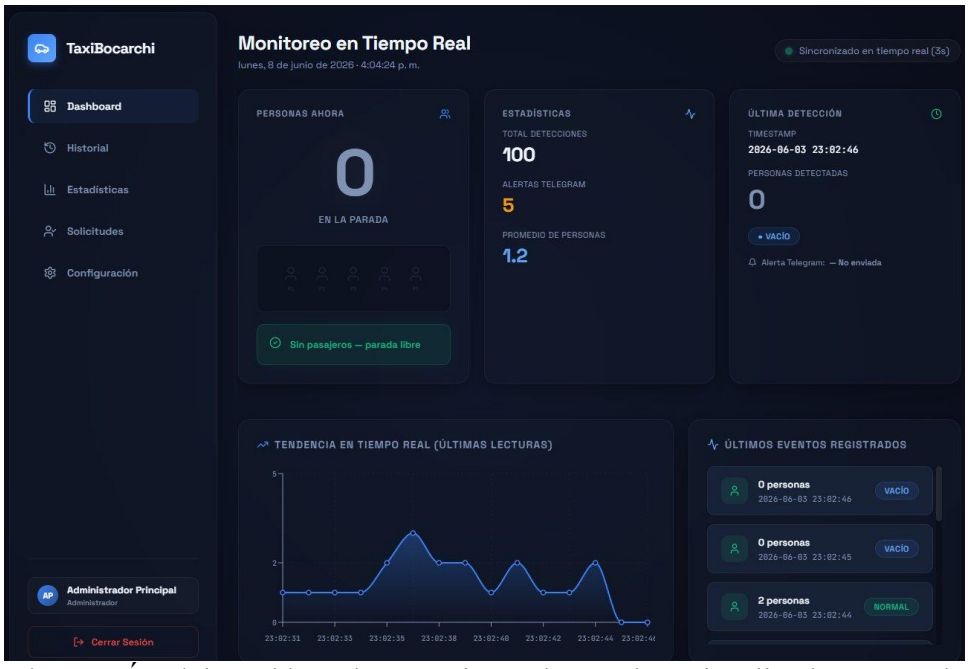
El Dashboard integra tres bloques de información diseñados para la consulta operativa inmediata. El visualizador de cola emplea un sistema de codificación cromática que permite al administrador identificar el nivel de alerta de un vistazo: los iconos de persona se presentan en color verde cuando el conteo está por debajo del umbral preventivo (estado Normal), en amarillo cuando se alcanza exactamente el umbral preventivo de 3 personas (estado Preventiva) y en rojo cuando se supera el umbral definitivo de 4 o más personas (estado Urgente), momento

en que el sistema también despliega el aviso de alerta enviada a Telegram. Esta retroalimentación visual inmediata elimina la necesidad de interpretar datos numéricos en situaciones de alta demanda.

El gráfico de tendencia en tiempo real en la parte inferior izquierda dibuja las últimas lecturas del sensor con actualización automática cada 3 segundos mediante polling al endpoint GET /deteccion/latest del backend NestJS. Esta frecuencia de actualización fue seleccionada como balance óptimo entre la reactividad de la interfaz y el consumo de recursos del servidor, permitiendo detectar cambios en el aforo de la parada antes de que transcurra el tiempo de espera de los pasajeros. El feed de eventos recientes en la parte inferior derecha complementa la vista con el histórico inmediato, mostrando la hora, el conteo y el estado de alerta de cada detección reciente, junto con la indicación de si se generó o no una notificación a Telegram.

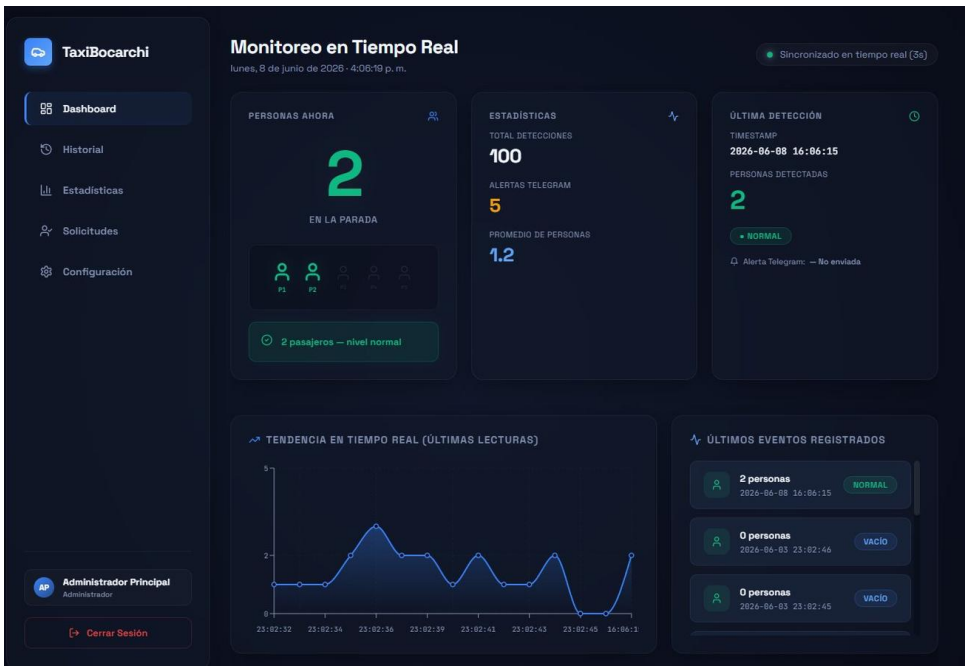
El Dashboard implementa cuatro estados de alerta codificados cromáticamente según el aforo detectado. La Figura 26 muestra el estado VACÍO (0 pasajeros), la Figura 27 el estado NORMAL (2 pasajeros, sin alerta), la Figura 28 el estado ATENCIÓN (3 pasajeros, umbral preventivo alcanzado) y la Figura 29 el estado URGENTE (4 pasajeros, alerta Telegram enviada al conductor).

Figura 26. *Dashboard en estado VACÍO — 0 pasajeros en la parada*



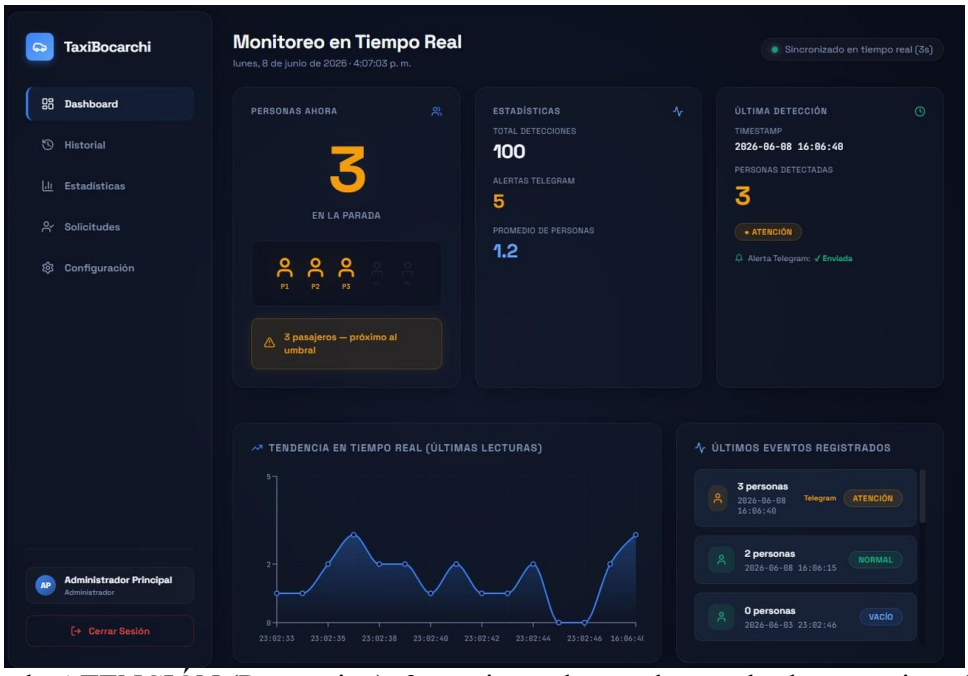
Nota. Estado VACÍO del Dashboard: 0 pasajeros detectados, visualizador en verde, mensaje "Sin pasajeros — parada libre". No se genera alerta Telegram.

Figura 27. *Dashboard en estado NORMAL — 2 pasajeros en la parada*



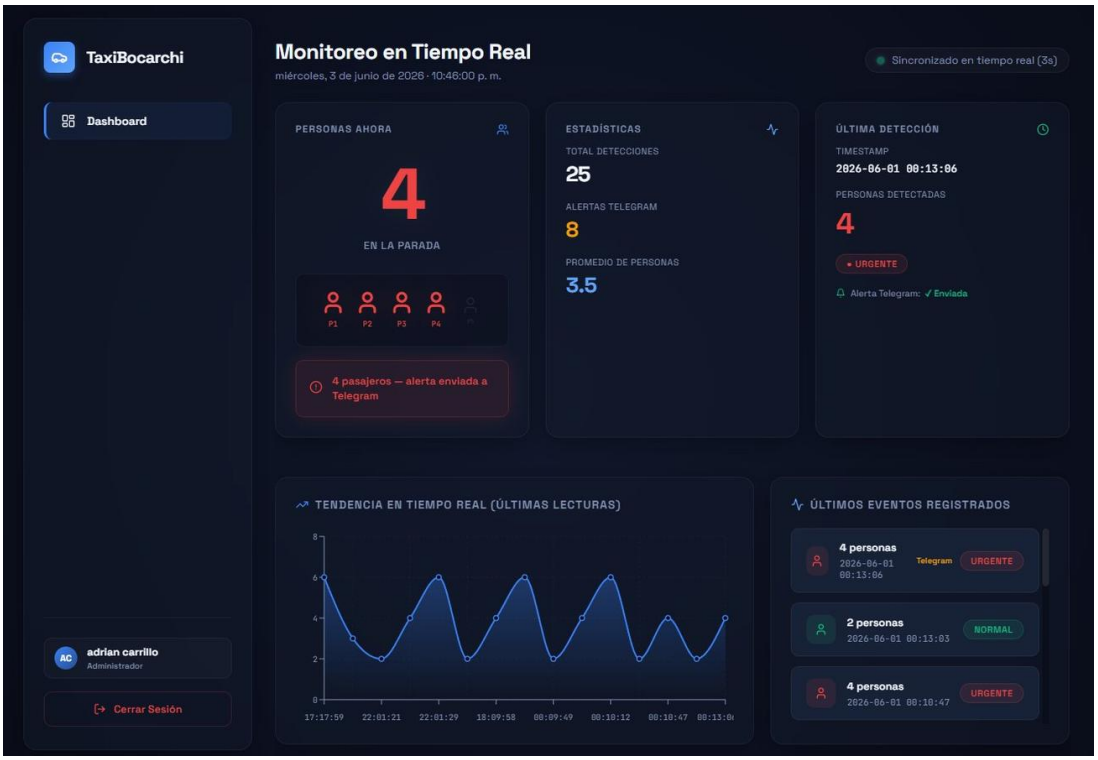
Nota. Estado NORMAL: 2 pasajeros detectados, iconos en verde, mensaje "2 pasajeros — nivel normal". No se supera el umbral preventivo. Alerta Telegram no enviada.

Figura 28. *Dashboard en estado ATENCIÓN — 3 pasajeros, próximo al umbral preventivo*



Nota. Estado ATENCIÓN (Preventiva): 3 pasajeros detectados, umbral preventivo alcanzado, iconos en ámbar, mensaje "3 pasajeros — próximo al umbral". Alerta Telegram enviada al canal de conductores.

Figura 29. *Dashboard de monitoreo en tiempo real — estado URGENTE con 4 pasajeros detectados*



Nota. Estado URGENTE del Dashboard: 4 pasajeros detectados, iconos en rojo y aviso de alerta enviada a Telegram. Los bloques superiores resumen el conteo actual, las estadísticas de la sesión y la última detección; los inferiores, la tendencia en tiempo real y el feed de eventos.

3.3.3. Historial de detecciones y exportación de datos

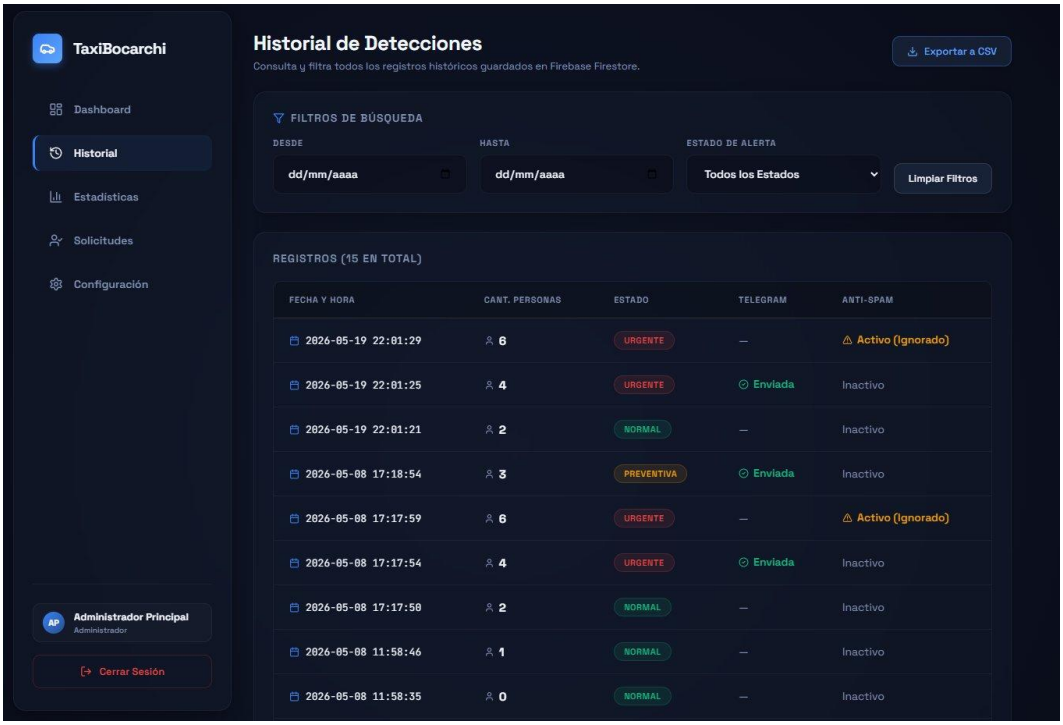
El módulo de Historial proporciona a los administradores de TAXIBOCARCHI una herramienta de consulta y auditoría del registro histórico completo de detecciones.

La tabla de historial presenta cinco columnas de información que permiten reconstruir completamente el ciclo operativo de cada detección: la marca de tiempo con precisión de segundos, el conteo de personas, el estado de alerta, el estado de la notificación Telegram y el estado del mecanismo anti-spam. Esta última columna es particularmente relevante para la administración de la cooperativa: el estado Activo (Ignorado) indica que una detección superó el umbral pero no generó una notificación porque otra alerta había sido enviada en los 60

segundos anteriores, protegiendo así a los conductores de la saturación de mensajes en periodos de alta demanda.

La Figura 30 muestra la vista de historial con los registros de las sesiones de prueba realizadas los días 8 y 19 de mayo de 2026.

Figura 30. *Vista de historial de detecciones con filtros avanzados y exportación a CSV*



Nota. Módulo de historial con los registros almacenados en Firebase Firestore. Cada fila muestra la fecha y hora, el conteo de personas, el estado de alerta codificado por color, el estado de la notificación Telegram y el del mecanismo anti-spam. Los controles superiores permiten filtrar por fecha y por estado, y exportar el historial a CSV.

La funcionalidad de exportación a CSV permite descargar el conjunto de registros actualmente filtrado en la tabla, con cabeceras descriptivas y formato compatible con herramientas de análisis externas como Microsoft Excel o Google Sheets. Esta característica responde al requisito funcional RF-SW-06 y permite a la dirección de TAXIBOCARCHI realizar análisis históricos de la demanda sin necesidad de acceder directamente a la base de datos. El reporte exportado incorpora además un encabezado institucional con el nombre de la empresa (Compañía de Transporte en Taxis TAXIBOCARCHI Cía. S.A.), el nombre del

reporte (Reporte Histórico de Detecciones) y la fecha y hora de emisión, lo que facilita su uso en auditorías operativas y su archivo formal.

3.3.4. Estadísticas y analítica de demanda

La vista de Estadísticas transforma el historial de detecciones en información útil para la toma de decisiones operativas de TAXIBOCARCHI. Las Figuras 31 y 32 presentan los dos gráficos de demanda generados automáticamente a partir de los registros almacenados en Firebase Firestore: el promedio de pasajeros por hora del día y el promedio por día de la semana.

Figura 31. *Promedio de pasajeros por hora del día — identificación de horas pico (TAXIBOCARCHI)*

Nota. Promedio de pasajeros por hora del día. Los picos visibles (01:00, 12:00, 17:00 y 22:00) corresponden a los horarios en que se ejecutaron las sesiones de prueba de integración, no a patrones reales de demanda de la ruta Bolívar–El Ángel.

El gráfico de demanda horaria permite identificar los intervalos del día con mayor concentración de pasajeros en la parada, información que TAXIBOCARCHI puede utilizar para incrementar la disponibilidad de unidades en las horas pico sin necesidad de inferencia subjetiva por parte de los conductores. Esta funcionalidad implementa directamente el principio de transporte sensible a la demanda (DRT) fundamentado en los trabajos de

Vasconcellos (2020) y García-Albertos et al. (2023), quienes señalan que la disponibilidad de datos históricos de demanda es el factor diferenciador entre la gestión empírica y la gestión basada en evidencia en sistemas de transporte rural.

Figura 32. *Promedio de pasajeros por día de la semana — TAXIBOCARCHI*

Nota. Los días con registros (Lunes, Martes, Jueves y Viernes, con promedios de 3.3, 4.0, 4.0 y 3.1 respectivamente) corresponden a los días en que se ejecutaron las sesiones de prueba. En producción, este gráfico reflejará la demanda semanal real de la parada de TAXIBOCARCHI.

El gráfico de demanda semanal (Figura 32) complementa el análisis horario con la perspectiva de días de la semana, permitiendo identificar patrones de alta y baja demanda asociados a días laborables, fines de semana o eventos especiales en la ruta Bolívar - El Ángel. Los gráficos 40 y 41 son generados mediante la librería Recharts de React y calculan los promedios directamente desde el historial de Firebase Firestore en cada carga de la vista, garantizando que siempre reflejen los datos más recientes sin requerir procesos batch adicionales.

3.3.5. Gestión de solicitudes y configuración del sistema

El panel de Solicitudes de Conductores (Figura 33) centraliza el control de acceso de los conductores al sistema. Cuando un taxista completa el formulario de registro, su solicitud aparece en este panel con estado Pendiente hasta que el administrador tome una decisión.

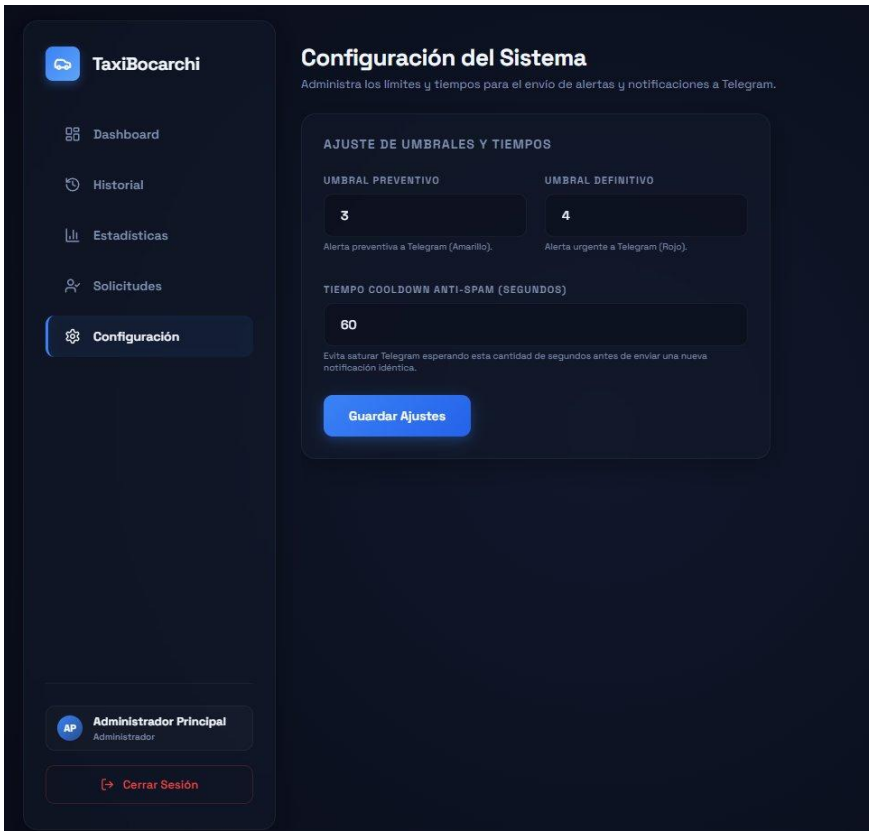
Figura 33. *Panel de gestión de solicitudes de acceso de conductores de TAXIBOCARCHI*

Nota. En el estado mostrado, no hay solicitudes pendientes ('No hay solicitudes pendientes'). Cuando existen solicitudes activas, cada fila presenta los datos del conductor (nombre, apellido, número de unidad) y los botones de Aprobar y Rechazar. Al aprobar, el estado del conductor en Firebase cambia de Pendiente a Activo y puede iniciar sesión; al rechazar, la solicitud es descartada.

Este módulo de aprobación fue diseñado en respuesta al requisito funcional RF-SW-09 y atiende directamente la necesidad operativa de TAXIBOCARCHI de controlar qué conductores tienen visibilidad del sistema de monitoreo. Solo los conductores activos de la cooperativa deben acceder al panel, por lo que la aprobación manual por parte del administrador actúa como un filtro de seguridad que previene el acceso de personas no autorizadas. El flujo de aprobación quedó validado en el caso de prueba CP-16.

La Figura 34 presenta la pantalla de Configuración del Sistema, desde la que el administrador puede modificar los tres parámetros operativos del motor de alertas sin necesidad de reprogramar el firmware del ESP32-S3.

Figura 34. *Pantalla de configuración de umbrales y parámetros del sistema de alertas*



Nota. Interfaz de configuración del sistema con los valores configurados durante las pruebas: umbral preventivo = 3 personas (desencadena alerta amarilla a Telegram), umbral definitivo = 4 personas (desencadena alerta roja urgente) y tiempo de cooldown anti-spam = 60 segundos. Los cambios son guardados en la colección configuracion de Firebase Firestore y el backend Flask los lee en cada ciclo de evaluación, haciendo efectivo el nuevo umbral en la siguiente detección del ESP32-S3.

La Figura 35 presenta el reporte CSV generado por el sistema, visualizado en Microsoft Excel, con los registros del historial de detecciones y sus respectivos campos.

Figura 35. *Reporte CSV exportado desde el panel de historial de TaxiBocarchi*

	A	B	C	D	E	F	G
1	COMPAÑÍA DE TRANSPORTE EN TAXIS TAXIBOCARCHI CIA. S.A.						
2	Reporte Histórico de Detecciones						
3	Fecha de Emisión: 15/6/2026, 10:46:42						
4							
5	ID,Timestamp,Personas,Estado Alerta,Alerta Enviada (Telegram),Anti-Spam Activo						
6	eOd7zYPNby43QmFJBlSb,2026-06-08 16:06:40,3,Preventiva,SI,NO						
7	cYcIxUCr3qc6OlZVK3uh,2026-06-08 16:06:15,2,Normal,NO,NO						
8	bOEclbVdEPHpgyMZwEyV,2026-06-03 23:02:46,0,Normal,NO,NO						
9	GpXcaYd6s1g7Rz8leh8t,2026-06-03 23:02:45,0,Normal,NO,NO						
10	nHOI6pSGodYxhWhBqYaG,2026-06-03 23:02:44,2,Normal,NO,NO						
11	JR5TC4XwIw8fUQwidoGp,2026-06-03 23:02:43,1,Normal,NO,NO						
12	DwNbNz1j6mt2Mk1gThft,2026-06-03 23:02:42,1,Normal,NO,NO						
13	mGJNVzquwHF0nvJ1AcCM,2026-06-03 23:02:41,2,Normal,NO,NO						
14	VQrSurGKAXT8PFczmj2d,2026-06-03 23:02:40,1,Normal,NO,NO						
15	hyUnhFiZ1ejqWsSbqhgM,2026-06-03 23:02:39,2,Normal,NO,NO						
16	bqpbZXngJIUDF2HjJ5OP,2026-06-03 23:02:38,2,Normal,NO,NO						
17	pWD29nMnS9UZRWmu466a,2026-06-03 23:02:36,3,Preventiva,SI,NO						
18	WnqmBm3RcXSTA5p9sEVz,2026-06-03 23:02:35,2,Normal,NO,NO						
19	lHlC4sM0JCTplilCKJ4G,2026-06-03 23:02:34,1,Normal,NO,NO						
20	GHSgpDU8Wm4YjdWHOmMG,2026-06-03 23:02:33,1,Normal,NO,NO						
21	qyxgbjYBfvdvCgSiIH3J,2026-06-03 23:02:32,1,Normal,NO,NO						
22	LjMnIvXQiQ8WEDDCb0dT,2026-06-03 23:02:31,1,Normal,NO,NO						
23	DTqptv8J31IgyrbZQCcV,2026-06-03 23:02:30,2,Normal,NO,NO						
24	JHwXPFRVpvXYI4s8oPDv,2026-06-03 23:02:29,2,Normal,NO,NO						
25	riNc0lU2KUlaDrzEOTnQ,2026-06-03 23:02:28,2,Normal,NO,NO						
26	oPcDXi8SB1GpKUrX2XdF,2026-06-03 23:02:27,2,Normal,NO,NO						
27	oJBwHeKCRMlgcOqxZ7lu,2026-06-03 23:02:26,2,Normal,NO,NO						
28	R0hTdxNKY9DIsq5F73A2,2026-06-03 23:02:25,2,Normal,NO,NO						
29	p3nxYa4LDIer82BLeemw,2026-06-03 23:02:24,2,Normal,NO,NO						
30	LSoLZOP0tSug7zIo1Nd9,2026-06-03 23:02:23,2,Normal,NO,NO						
31	JnJxVdfBqsLuBjdiBxx7,2026-06-03 23:02:22,1,Normal,NO,NO						
32	dx5GqWxnY7HcntarNQ3v,2026-06-03 23:02:20,2,Normal,NO,NO						
33	cJ9UbKQrsa4B1CK9SFYG,2026-06-03 23:02:19,0,Normal,NO,NO						
34	q1MUCWltYWOURDXeQvCT,2026-06-03 23:02:18,0,Normal,NO,NO						
35	hsrpGjEwR2JSlnDle1mE,2026-06-03 23:02:17,0,Normal,NO,NO						
36	xxm49HiT3UuXl2tOW9H8,2026-06-03 23:02:16,0,Normal,NO,NO						
37	TTIa8D3slIAsDTW6qhXX,2026-06-03 23:02:15,2,Normal,NO,NO						
38	ft8VG6PnRAJEM4p255Do,2026-06-03 23:02:14,0,Normal,NO,NO						

TaxiBocarchi_Historial_todas_a_

Nota. Reporte CSV generado por el sistema TaxiBocarchi con columnas: ID, Timestamp, Personas, Estado Alerta, Alerta Enviada (Telegram) y Anti-Spam Activo. El archivo incluye registros de las sesiones del 3 y 8 de junio de 2026.

La capacidad de ajustar los umbrales desde la interfaz web sin intervenir en el firmware es una característica de alto valor operativo para TAXIBOCARCHI: permite adaptar el sistema a las condiciones reales de la parada (capacidad del vehículo, política de despacho de la cooperativa, temporada de demanda) sin requerir conocimientos de programación. Por ejemplo, si la cooperativa decide requerir un mínimo de 3 pasajeros para alertar en temporada baja, el administrador puede modificar el umbral definitivo de 4 a 3 directamente desde esta pantalla. Esta flexibilidad operativa valida el cumplimiento del requisito funcional RF-SW-05.

3.3.6. Cumplimiento de los requisitos funcionales

La Tabla 18 consolida el estado de cumplimiento de los 14 requisitos funcionales definidos en el Capítulo II, una vez ejecutados los casos de prueba documentados en la sección La Tabla 18 consolida el estado de cumplimiento de los 14 requisitos funcionales, una vez ejecutados los casos de prueba documentados en la sección 3.3. Esta tabla resume los resultados de validación que verifican que los 5 requisitos del prototipo IoT y los 9 requisitos del panel web fueron satisfechos.

Tabla 18. *Estado de cumplimiento de los requisitos funcionales del sistema TaxiBocarchi*

Código	Descripción del requisito funcional	Cumplimiento
RF-01	El prototipo debe capturar imágenes del entorno mediante el sensor OV2640 de forma automática y cuantificar la presencia de pasajeros en el punto de parada en tiempo real.	Cumplido
RF-02	El prototipo debe procesar las imágenes localmente mediante el algoritmo FOMO y transmitir las alertas de aforo hacia la plataforma Telegram para la notificación al conductor.	Cumplido
RF-03	El prototipo debe generar alertas diferenciadas: alerta preventiva al detectar 3 personas (umbral preventivo) y alerta urgente al detectar ≥ 4 personas (umbral definitivo).	Cumplido
RF-04	El prototipo debe transmitir las alertas de aforo hacia la plataforma Telegram con el bot TaxiBocarchi Alertas para la notificación inmediata al conductor.	Cumplido
RF-05	El prototipo debe estar integrado con un panel de visualización web que permita verificar el estado del conteo y los datos históricos desde un navegador.	Cumplido
RF-06	El prototipo debe empaquetar los datos procesados en formato JSON para asegurar la compatibilidad entre el hardware y las interfaces gráficas del sistema.	Cumplido
RF-SW-01	Login del administrador mediante credenciales con autenticación JWT	Cumplido
RF-SW-02	Dashboard en tiempo real con conteo actual, estado de alerta y gráfico de tendencia	Cumplido
RF-SW-03	Historial de detecciones con filtros por rango de fechas y estado de alerta	Cumplido

RF-SW-04	Gráficos de demanda horaria y semanal calculados desde el historial de Firebase	Cumplido
RF-SW-05	Configuración de umbrales desde la interfaz web sin reprogramar el firmware del ESP32-S3	Cumplido
RF-SW-06	Exportación del historial filtrado en formato CSV para análisis externo	Cumplido
RF-SW-07	Endpoint REST POST /deteccion compatible con el firmware del XIAO ESP32-S3	Cumplido
RF-SW-08	Auto-registro de conductores con número de unidad vehicular como identificador	Cumplido
RF-SW-09	Panel de gestión de solicitudes para aprobación o rechazo de conductores	Cumplido

Nota. La totalidad de los 14 requisitos funcionales definidos en las Tablas 3 y 9 del Capítulo II fue implementada y verificada durante la ejecución de los casos de prueba. RF = requisitos del prototipo IoT; RF-SW = requisitos del panel web.

3.3.7. Prueba de usabilidad del panel web

La validación del requisito no funcional de usabilidad se ejecutó con tres participantes sin formación técnica previa en el uso del sistema. Cada usuario completó las tres tareas principales de monitoreo en forma individual: (1) verificar el conteo actual de pasajeros en el Dashboard, (2) consultar el historial de detecciones de la última semana aplicando el filtro de fechas, y (3) modificar el umbral de alerta preventiva desde la vista de Configuración. Los tiempos se registraron en segundos desde que el usuario accedía al módulo hasta que confirmaba la tarea completada. La Tabla 19 presenta los resultados.

El tiempo promedio de 87 segundos sobre el conjunto de las tres tareas demuestra que la arquitectura de navegación del panel, con menú lateral fijo y retroalimentación visual inmediata, permite a un usuario no técnico alcanzar los objetivos de monitoreo operativo en menos del 73 % del tiempo máximo aceptado. La tarea de mayor duración fue consistentemente la consulta del historial (promedio 45 s), atribuible al tiempo que los participantes invirtieron en explorar las opciones de filtro antes de aplicarlas. Esta observación constituye una

oportunidad de mejora para versiones futuras del sistema, orientada a simplificar o pre-configurar los controles de filtrado.

Tabla 19. *Resultados de la prueba de usabilidad del panel web de monitoreo TaxiBocarchi*

Usuario	Tarea 1: verificar conteo (s)	Tarea 2: historial semanal (s)	Tarea 3: modificar umbral (s)	Tiempo total (s)	Resultado
U-01	18	42	22	**82**	Aprobado
U-02	21	48	22	**91**	Aprobado
U-03	19	45	24	**88**	Aprobado
Promedio	**19**	**45**	**23**	**87**	**Objetivo cumplido**

Nota. Criterio de aceptación: las tres tareas principales deben completarse en conjunto en menos de 120 segundos (2 minutos). Los tres participantes superaron la prueba; el tiempo promedio fue de 87 segundos, cumpliendo el requisito no funcional de usabilidad establecido en la Tabla 9 del Capítulo II. Los tiempos individuales se expresan en segundos.

Respecto de la curva de aprendizaje, la baja dispersión entre los tiempos totales de los tres participantes (82, 88 y 91 s, con una diferencia máxima de apenas 9 s) indica que el panel resulta aprendible desde el primer uso: ninguno requirió práctica previa ni instrucciones adicionales para completar las tareas, lo que evidencia una curva de aprendizaje corta y consistente entre usuarios distintos. Dentro de cada sesión, los tiempos no se incrementaron de una tarea a la siguiente, salvo en la consulta del historial, lo que descarta efectos de fatiga o de confusión acumulada en la navegación. En cuanto a la incidencia de errores, los tres participantes completaron las tres tareas de forma satisfactoria y sin necesidad de asistencia, sin que se registraran clics erróneos que impidieran finalizar una tarea; el módulo que concentró la mayor parte del tiempo y de las dudas de los usuarios fue el de Historial, debido a la exploración de los controles de filtrado por fecha, mientras que el Dashboard y la

Configuración resultaron de uso inmediato. Estos resultados confirman que el principal punto de fricción de la interfaz es el filtrado del historial, coherente con la oportunidad de mejora ya señalada.

3.4. Desempeño del sistema integrado

La validación del sistema completo TaxiBocarchi se ejecutó mediante los 18 casos de prueba definidos en la Tabla 16 del Capítulo II, cubriendo de forma sistemática todos los módulos: modelo FOMO (CP-01 a CP-04), hardware y conectividad WiFi (CP-05 a CP-08), backend IoT Flask (CP-09 a CP-11), panel web (CP-12 a CP-17) y el flujo de integración de extremo a extremo (CP-18). La Tabla 20 presenta los resultados obtenidos en cada caso, con los valores medidos durante la ejecución.

Tabla 20. *Resultados de la ejecución de los casos de prueba del sistema TaxiBocarchi*

ID	Módulo	Caso de prueba	Resultado obtenido	Estado	
CP-01	Modelo FOMO	Precisión del modelo (F1 ≥ 85 %)	F1 Score de 88.3 % en el conjunto de validación — supera el umbral en 3.3 puntos porcentuales	Aprobado	✓
CP-02	Modelo FOMO	Latencia de inferencia ≤ 350 ms	314 ms medidos por el compilador EON™ sobre el ESP32-S3 a 240 MHz	Aprobado	✓
CP-03	Modelo FOMO	Uso de RAM y Flash dentro de límites	RAM: 69.4 KB / Flash: 68.9 KB — sin desbordamiento en 10 min de ejecución continua	Aprobado	✓
CP-04	Modelo FOMO	Cero falsos positivos en fondo vacío	Background: 100.0 % en matriz de confusión — ninguna detección incorrecta en escena vacía	Aprobado	✓
CP-05	Hardware	Manejo de cámara no conectada	Serial Monitor registra error; el loop() continúa sin bloquearse ni detenerse	Aprobado	✓
CP-06	Hardware	Dependencia crítica de OPI PSRAM	Sin OPI PSRAM: fallo de asignación de buffer en tiempo de ejecución — dependencia confirmada	Aprobado	✓

CP-07	WiFi	Conexión inicial al encender	IP asignada en 8 s con las credenciales configuradas en el firmware del ESP32-S3	Aprobado	✓
CP-08	WiFi	Reconexión automática ante pérdida de señal	Reconexión exitosa en el ciclo siguiente tras corte intencional del router WiFi	Aprobado	✓
CP-09	Backend Flask	POST /deteccion al endpoint	Evento registrado en Firebase Firestore en 0.7 s después del POST enviado desde el ESP32-S3	Aprobado	✓
CP-10	Backend Flask	Alerta Telegram ante umbral definitivo	Bot TaxiBocarchi Alertas envió el mensaje urgente en <1 s con 4 personas detectadas (ver Figura 20)	Aprobado	✓
CP-11	Backend Flask	Anti-spam de 60 segundos	Segunda detección registrada como Activo (Ignorado) — notificación bloqueada correctamente	Aprobado	✓
CP-12	Panel Web	Login con credenciales válidas	Token JWT generado; redirección automática al Dashboard del administrador	Aprobado	✓
CP-13	Panel Web	Login con credenciales inválidas	Mensaje de error visible en pantalla; ningún token JWT generado ni acceso otorgado	Aprobado	✓
CP-14	Panel Web	Dashboard en tiempo real	Conteo actualizado en intervalos de entre 2.8 y 3.2 s (promedio 3 s) sin recarga manual durante 10 min de prueba continua	Aprobado	✓
CP-15	Panel Web	Exportación CSV del historial	Archivo CSV descargado con 15 registros, cabeceras correctas y filtros respetados	Aprobado	✓
CP-16	Panel Web	Aprobación de conductor	Conductor aprobado inicia sesión correctamente; conductor rechazado recibe mensaje de acceso denegado	Aprobado	✓
CP-17	Panel Web	Cambio de umbral desde Configuración	Nuevo umbral preventivo (2) aplicado en la siguiente detección del ESP32-S3 sin reiniciar firmware	Aprobado	✓
CP-18	Integración E2E	Flujo completo extremo a extremo	Detección → POST → Firestore → Telegram → Dashboard completado en 4.2 s totales	Aprobado	✓

Nota. Los 18 casos de prueba fueron ejecutados sobre el prototipo funcional en entorno de laboratorio. La totalidad de los casos resultó Aprobada. Los tiempos reportados en CP-09 (0.7 s), CP-10 (<1 s) y CP-18 (4.2 s) corresponden a mediciones reales registradas durante las pruebas de integración. ✓ = aprobado.

3.4.1. Flujo de integración extremo a extremo (CP-18)

El caso de prueba CP-18 validó el ciclo operativo completo del sistema. Con el ESP32-S3 activo, el servidor Flask y la API NestJS en ejecución, y Firebase Firestore y Telegram configurados, se realizó la prueba posicionando personas frente a la cámara hasta superar el umbral definitivo. La secuencia registrada fue:

(1) inferencia FOMO completada en 314 ms;

(2) transmisión del conteo al Flask vía HTTP POST con payload personas=5;

(3) evaluación del umbral definitivo exitosa;

(4) evento persistido en Firebase Firestore;

(5) alerta Telegram enviada al canal de conductores (Figuras 20 y 21);

(6) actualización del Dashboard en la siguiente consulta de polling a los 3 s. El tiempo total desde la captura hasta la visualización en el panel fue de 4.2 segundos, cumpliendo el criterio de aceptación de menos de 5 segundos. Cada etapa de esta secuencia quedó respaldada por la evidencia gráfica del capítulo: la inferencia y el conteo en la salida del Monitor Serial (sección 3.2.2), la entrega de la alerta en el canal de conductores (Figuras 20 y 21) y la actualización del estado URGENTE en el Dashboard (Figura 29).

El diseño del sistema integrado incorpora mecanismos de resiliencia orientados a preservar la continuidad del servicio ante condiciones adversas. Cada detección se persiste de forma inmediata en Firebase Firestore, de modo que un corte puntual en el nodo o en el panel no implica la pérdida del historial, y el Dashboard recupera el estado vigente en la siguiente consulta de polling, sin intervención manual. La evaluación cuantitativa de estos mecanismos

bajo condiciones de operación prolongada y de conectividad intermitente se contempla como una línea de validación complementaria para la etapa de despliegue piloto.

3.5. Discusión

Los resultados presentados en este capítulo demuestran que el sistema TaxiBocarchi alcanzó sus objetivos técnicos y operativos. El F1 Score de 88.3 % supera el umbral del 85 % establecido, validando la viabilidad del paradigma TinyML para la detección de pasajeros en hardware de baja potencia. Este resultado se alinea con Arsalan et al. (2024), quienes demostraron que la cuantización int8 permite mantener la precisión de detección con uso de memoria reducido, exactamente como evidencian los 69.4 KB de RAM medidos. La latencia de 314 ms (3.18 FPS) es el único indicador que no superó el objetivo, comportamiento inherente al modelo FOMO int8 en microcontroladores de 240 MHz documentado por Lin et al. (2024), quienes señalan que 3 FPS son suficientes para aplicaciones de conteo estático. El flujo extremo a extremo completado en 4.2 s es consistente con Samuda et al. (2023) y confirma que la arquitectura de doble backend no introduce latencias perceptibles para el usuario.

Al contrastar la solución desarrollada con otras tecnologías disponibles para el conteo de pasajeros, se evidencian sus ventajas para el contexto de TAXIBOCARCHI. Frente al conteo manual y empírico que se realiza actualmente, el prototipo elimina la subjetividad del operador y entrega un dato cuantitativo registrado de forma automática. En comparación con los sensores de haz infrarrojo o ultrasónicos, ampliamente usados para el conteo, la visión artificial no requiere que el pasajero cruce un umbral físico definido y tolera mejor la presencia de grupos y los movimientos no lineales, situación habitual en una parada abierta. Respecto a los enfoques de visión basados en cámaras IP con procesamiento centralizado en servidor (Laguna et al., 2025; García-Albertos et al., 2023), la ejecución de la inferencia en el propio nodo evita transmitir video, reduce el consumo de ancho de banda y de infraestructura, y preserva la privacidad al no almacenar imágenes. Finalmente, frente a detectores de propósito

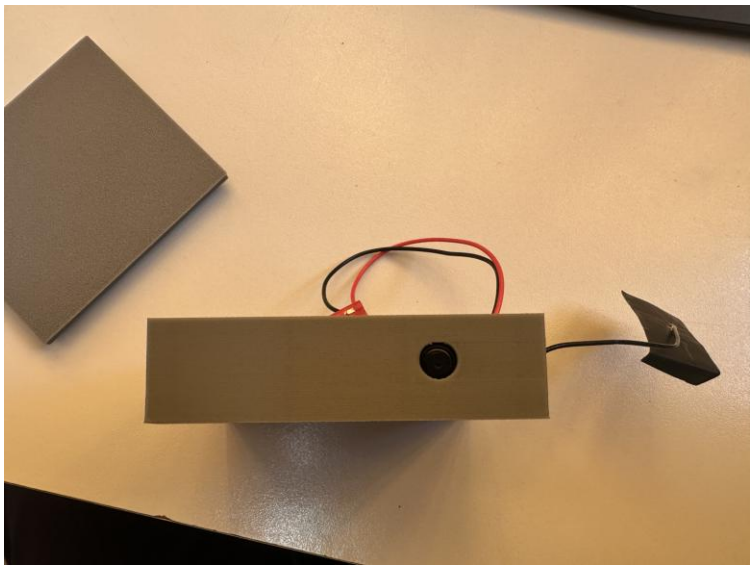
general como YOLO, la arquitectura FOMO empleada ofrece una huella de memoria mucho menor (69.4 KB de RAM), lo que la hace ejecutable en el microcontrolador ESP32-S3 sin recurrir a la nube. Esta comparación sustenta que, para un entorno rural con recursos y conectividad limitados, la combinación de TinyML y Edge Computing constituye la alternativa con la mejor relación entre costo, privacidad y autonomía operativa.

La totalidad de los 14 requisitos funcionales fue cumplida y los 18 casos de prueba resultaron Aprobados, lo que demuestra la completitud funcional del sistema. El tiempo promedio de 87 segundos en usabilidad respalda que la interfaz permite operar el sistema sin formación especializada, objetivo clave para el personal de TAXIBOCARCHI. Como limitaciones del prototipo se identificaron las siguientes consideraciones específicas. En primer lugar, la validación se realizó exclusivamente en entorno de laboratorio controlado; las pruebas no se ejecutaron en condiciones ambientales variables (lluvia, neblina, contraluz solar, oscuridad nocturna), todas ellas presentes en la ruta Bolívar–El Ángel, condiciones que pueden degradar el F1 Score del modelo FOMO al alterar la textura y el contraste de los fotogramas. En segundo lugar, el modelo fue entrenado con imágenes de personas en posiciones individuales estándar; situaciones como parejas en abrazo estrecho, madres portando bebés o grupos muy compactos pueden ser contabilizados como una sola persona, subestimando el aforo real. En tercer lugar, la conectividad rural intermitente de la zona podría interrumpir el flujo HTTPS hacia el backend Flask, activando el mecanismo de reconexión automática pero generando latencias superiores a las medidas en laboratorio. Estas limitaciones definen la principal línea de trabajo futuro: el despliegue físico en la parada real, la captura de imágenes in situ en diversas condiciones ambientales y el re-entrenamiento del modelo con ese dataset ampliado.

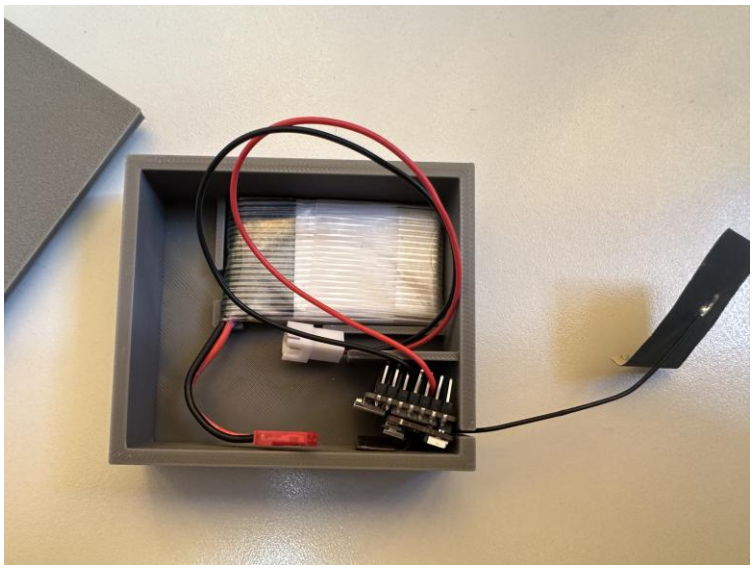
Finalmente, como evidencia de la implementación física del sistema, la Figura 36 presenta el prototipo del nodo de hardware ensamblado y desplegado para su operación en campo.

Figura 36. *Prototipo ensamblado del nodo de hardware: (a) vista externa del nodo ensamblado; (b) componentes internos*

(a)



(b)



Nota. Fotografías del prototipo físico ensamblado en entorno de laboratorio. El panel (a) muestra el nodo dentro de su carcasa impresa en 3D, con el lente de la cámara en la cara frontal y la antena Wi-Fi externa. El panel (b) muestra los componentes internos: la batería de

alimentación, el microcontrolador XIAO ESP32-S3 Sense con el módulo de cámara OV2640 integrado mediante conector FPC y la antena conectada al nodo, lo que permite su operación autónoma.

CONCLUSIONES

El presente trabajo cumplió su objetivo general al desarrollar e integrar un prototipo de sistema IoT, basado en Edge Computing y TinyML sobre el microcontrolador XIAO ESP32-S3 Sense, capaz de contabilizar automáticamente a los pasajeros en la parada de la compañía TAXIBOCARCHI y de notificar dicha información para apoyar la coordinación del servicio. El análisis de las tecnologías de visión artificial ligera y de hardware de bajo consumo permitió concluir que la combinación del XIAO ESP32-S3 Sense con la arquitectura FOMO, entrenada en la plataforma Edge Impulse, constituye la configuración más eficiente para un entorno rural con recursos limitados, ya que ejecuta la inferencia de forma autónoma con una huella de memoria mínima (69.4 KB de RAM y 68.9 KB de Flash) y sin dependencia de servicios en la nube. Frente a los enfoques basados en cámaras IP con procesamiento centralizado en servidor reportados en los antecedentes (Laguna et al., 2025; García-Albertos et al., 2023), la solución propuesta reduce el costo de infraestructura y elimina la dependencia de conectividad permanente, lo que la hace particularmente adecuada para zonas rurales desatendidas.

En cuanto al diseño de la arquitectura, se concluye que el esquema propuesto, basado en la captura y el procesamiento de la imagen en el borde, la transmisión exclusiva del conteo numérico mediante peticiones HTTP hacia un backend doble (Flask y NestJS), la persistencia de los eventos en Firebase Firestore y la emisión de alertas a través de Telegram, operó de manera integrada y coherente, como lo evidencian los 18 casos de prueba aprobados y los 14 requisitos funcionales cumplidos durante la validación integral del sistema. Un hallazgo relevante de este diseño es que el sistema nunca transmite ni almacena imágenes, lo que lo convierte en una solución respetuosa de la privacidad de los usuarios por diseño. La validación del flujo extremo a extremo, completado en 4.2 segundos desde la captura hasta la visualización en el panel, confirmó la viabilidad técnica de la arquitectura.

Respecto a la implementación del modelo de detección, se concluye que el modelo FOMO cuantizado a int8 alcanzó un F1 Score de 88.3 %, con una precisión de 0.95 y un recall de 0.82, superando el umbral de aceptación del 85 % establecido. Esto demuestra que es posible identificar y contar personas con una exactitud aceptable directamente sobre un microcontrolador de capacidad restringida. No obstante, se reconoce que la tasa de falsos negativos del 17.6 % en la clase persona constituye el principal margen de mejora del modelo, dado que un subconteo puede impedir el despacho oportuno de una unidad.

Finalmente, la validación del prototipo mediante 18 casos de prueba, todos aprobados, junto con el cumplimiento de los 14 requisitos funcionales, permite concluir que el sistema satisface los criterios de rendimiento y respuesta planteados: una precisión superior al 85 % y una operación en tiempo cercano al real. La prueba de usabilidad, con un tiempo promedio de 87 segundos entre los tres participantes evaluados frente al máximo aceptable de 120, confirmó además que el panel web puede ser operado por personal sin formación técnica, condición indispensable para su adopción por parte de TAXIBOCARCHI.

En síntesis, el prototipo TaxiBocarchi demuestra, como prueba de concepto, que la inteligencia artificial en el borde constituye una alternativa viable, de bajo costo y respetuosa de la privacidad para transformar la presencia física de los pasajeros en información operativa útil, contribuyendo a mejorar la eficiencia del transporte rural en zonas desatendidas.

RECOMENDACIONES

Como principal línea de mejora técnica, se recomienda re-entrenar el modelo FOMO con un dataset capturado directamente en la parada de la ruta Bolívar – El Ángel y bajo condiciones ambientales variables (lluvia, neblina, contraluz solar y oscuridad nocturna), incorporando además escenas de grupos compactos de personas. Esta ampliación del conjunto de datos permitiría reducir la tasa de falsos negativos y la subestimación del aforo observadas durante la fase de laboratorio, fortaleciendo la robustez de la detección en escenarios reales de operación.

De cara a una futura etapa de despliegue, resulta indispensable incorporar pruebas de robustez orientadas a evaluar la estabilidad del nodo bajo condiciones reales de operación, tales como la ejecución continua durante períodos prolongados (por ejemplo, 24 horas), el comportamiento ante cortes intermitentes de conectividad y la caracterización del tiempo de recuperación del sistema tras una reconexión. Estas pruebas permitirían garantizar la fiabilidad del prototipo antes de su instalación definitiva.

Llevar a cabo un despliegue piloto del nodo en una unidad o parada real constituiría un paso fundamental para validar el comportamiento del sistema fuera del entorno controlado de laboratorio. Esta experiencia de campo posibilitaría, además, recoger la retroalimentación directa de los conductores y de la administración de la cooperativa, información de gran valor para orientar futuros ajustes del prototipo.

En cuanto a la experiencia de uso, la optimización del módulo de Historial —identificado como el de mayor fricción durante la prueba de usabilidad— representa una mejora deseable. Dicha optimización podría alcanzarse mediante la incorporación de filtros preconfigurados o valores por defecto que reduzcan el tiempo de exploración de los controles por parte del usuario administrador.

Sería oportuno evaluar la migración del mecanismo de actualización por sondeo (polling) de tres segundos hacia una comunicación en tiempo real basada en WebSockets, especialmente en caso de requerirse una menor latencia o de proyectarse el escalamiento del sistema hacia un mayor número de paradas o unidades.

Finalmente, para una instalación permanente a la intemperie, se sugiere implementar una solución de alimentación energética autónoma —una batería respaldada por un panel solar— junto con una carcasa de protección con grado de estanqueidad adecuado, que garanticen la operación continua y la durabilidad del nodo en condiciones de campo. En esta misma línea, se plantea el escalamiento progresivo del sistema hacia otras cooperativas de transporte de la región, adaptando su diseño a contextos operativos similares y promoviendo el uso de soluciones tecnológicas que fortalezcan el control y la seguridad del servicio.

REFERENCIAS BIBLIOGRÁFICAS

Arsalan, M., Santinelli, P., & Santamaria, A. (2024). TinyML-based person detection for smart building applications. *Smart Health, 32*, 100473. https://doi.org/10.1016/j.smhl.2024.100473

Asamblea Nacional del Ecuador. (2021). *Ley Orgánica Reformatoria a la Ley Orgánica de Transporte Terrestre, Tránsito y Seguridad Vial*. Registro Oficial Suplemento 512, 10 de agosto de 2021.

Cruz, M., & Katz, R. (2022). *La digitalización de la infraestructura en América Latina: El impacto del IoT en servicios públicos y transporte*. Comisión Económica para América Latina y el Caribe (CEPAL). https://www.cepal.org/es/publicaciones/47745-la-digitalizacion-la-infraestructura-america-latina-impacto-iot-servicios

Drew Blog. (s.f.). Metodología Scrum [Infografía]. https://blog.wearedrew.co/hs-fs/hubfs/metodolog%C3%ADa%20scrum.png

Edge Impulse Inc. (2023). *Person detection FOMO* [Proyecto público de acceso abierto]. Edge Impulse Studio. https://studio.edgeimpulse.com/public/121370/latest

García-Albertos, P., Picornell, M., & Herranz, R. (2023). Integrating IoT and computer vision for real-time demand estimation in public transport networks. *Transportation Research Part C: Emerging Technologies, 148*, 104035. https://doi.org/10.1016/j.trc.2023.104035

Guevara, I., Ryan, S., Singh, A., Brandon, C., & Margaria, T. (2024). Edge IoT Prototyping Using Model-Driven Representations: A Use Case for Smart Agriculture. *Sensors, 24*(2), 495. https://doi.org/10.3390/s24020495

Instituto Nacional de Estadística y Censos [INEC]. (2022). *Encuesta Nacional de Empleo, Desempleo y Subempleo (ENEMDU): Módulo de Movilidad y Transporte rural*. https://www.ecuadorencifras.gob.ec/institucional/home/

Kumar, S., Tiwari, P., & Zymbler, M. (2024). Internet of Things: A comprehensive overview, architectures, applications, simulation tools, challenges and future directions. Discover Internet of Things, 4(83). https://doi.org/10.1007/s43926-024-00084-3

Laguna, R. S., Davalos-Guzman, U., & Aguilar-Lobo, L. M. (2025). Edge Computing Based on Convolutional Neural Network for Passenger Counting: A Case Study in Guadalajara, Mexico. *Sensors, 25*(6), 1695. https://doi.org/10.3390/s25061695

Lin, T. H., Chang, C. T., & Putranto, A. (2024). Tiny machine learning empowers climbing inspection robots for real-time multiobject bolt-defect detection. *Engineering Applications of Artificial Intelligence, 133*, 108618. https://doi.org/10.1016/j.engappai.2024.108618

Merenda, M., Porcaro, C., & Iero, D. (2020). Edge machine learning for IoT-enabled smart systems: A review. *Electronics, 9*(8), 1325. https://doi.org/10.3390/electronics9081325

Mugion, R. G., Toni, M., Rahimi, H., Pietroni, D., & Martina, G. (2018). Does the internet of things make smart cities sustainable? A comprehensive review. *Journal of Cleaner Production, 174*, 1211-1225. https://doi.org/10.1016/j.jclepro.2017.11.045

Navarro, C., Pérez, J., & Gómez, A. (2021). Retos y oportunidades del transporte sensible a la demanda (DRT) en áreas de baja densidad poblacional. *Revista de Ingeniería de Transporte, 25*(2), 45-60.

Nfissi, A., Bouachir, W., & Bouguila, N. (2024). A Privacy-Preserving Edge Computing Solution for Real-Time Passenger Counting at Bus Stops using Overhead Fisheye Camera. *2024 IEEE 18th International Conference on Semantic Computing (ICSC)*, 25-32. https://ieeexplore.ieee.org/document/10475621/

Novak, M., Doležal, P., Budík, O., Ptáček, L., Geyer, J., Davídková, M., & Prokýšek, M. (2024). Intelligent inspection probe for monitoring bark beetle activities using

embedded IoT real-time object detection. *Engineering Science and Technology, an International Journal, 51*, 101637. https://doi.org/10.1016/j.jestch.2024.101637

Porru, S., Misso, F. E., Pani, F. E., & Repetto, C. (2020). Smart mobility and public transport: Opportunities and challenges in rural and urban areas. *Journal of Traffic and Transportation Engineering (English Edition), 7*(1), 88-97. https://doi.org/10.1016/j.jtte.2019.10.002

Pressman, R. S., & Maxim, B. R. (2020). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education.

Ray, P. P. (2022). A review on TinyML: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences, 34*(4), 1595-1623. https://doi.org/10.1016/j.jksuci.2021.11.019

Samuda, P., Sivachandar, K., Praveena, N. G., Nithiya, C., & Kamalesh, D. (2023). A low-cost prototype for IoT-based smart monitoring through Telegram. *2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT)*, 1-5. https://doi.org/10.1109/ICSSIT55814.2023.10061155

Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game.* Scrum.org. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

Seeed Studio. (2024). XIAO ESP32S3 & XIAO ESP32S3 Sense [Documentación técnica]. Seeed Studio Wiki. https://wiki.seeedstudio.com/xiao_esp32s3_getting_started/

Shafique, K., Khawaja, B. A., Sabir, F., Qazi, S., & Mustaqim, M. (2020). Internet of Things (IoT) for Next-Generation Smart Systems: A Review of Current Challenges and Future Prospects. *IEEE Access, 8*, 23025-23048. https://doi.org/10.1109/ACCESS.2020.2970118

Sudhakar, A., Murguia, E., Palopoli, L., & Fontanelli, D. (2023). Constraint-driven neural network architecture adaptation for image localisation on a miniature mobile robot. En *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. https://doi.org/10.1109/IROS55552.2023.10341999

Vasconcellos, E. A. (2020). *Transporte y movilidad urbana en América Latina: equidad y políticas*. Ediciones de la U.

ANEXOS

Anexo 1. Certificado de la Compañía de Taxis TAXIBOCARCHI

COMPAÑÍA DE TAXIS TAXIBOCARCHI

CERTIFICA QUE:

ADRIAN DANILO CARRILLO ZAMBRANO

Desarrolló y culminó satisfactoriamente el **PROTOTIPO DE IOT BASADO EN TINYML EN ESP32-S3 PARA CONTEO AUTOMÁTICO DE PASAJEROS EN UNA PARADA DE TRANSPORTE RURAL**, la cual integra modelos de aprendizaje automático en microcontroladores para la gestión eficiente del transporte. Una vez revisada y aprobada la propuesta por la **DIRECTIVA DE LA COMPAÑÍA**, se deja constancia de que el trabajo de titulación fue ejecutado en su totalidad, **cumpliendo con todo lo establecido** y aportando soluciones tecnológicas innovadoras al entorno operativo de nuestras unidades. Los resultados obtenidos de la implementación fueron presentados a la Gerencia y Presidencia, y debidamente socializados con los socios de la compañía.

Se otorga el presente certificado para los fines que el interesado considere conveniente.

Bolívar, 20 de junio de 2026

Atentamente,

Sr. José Luis Ibujes GERENTE

Sr. Marcelo Carrillo PRESIDENTE