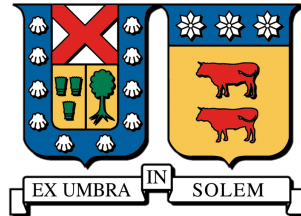


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTRÓNICA Y TELEMÁTICA

VALPARAÍSO – CHILE



**SISTEMA INTELIGENTE DE MONITOREO
MULTICÁMARA
MEDIANTE VISIÓN POR COMPUTADOR E
INTELIGENCIA ARTIFICIAL**

BENJAMÍN IGNACIO RAMÍREZ BARRIL

**MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL TELEMÁTICO**

PROFESOR GUÍA: MARCOS ZÚÑIGA BARRAZA
PROFESOR CORREFERENTE: NICOLÁS JARA CARVALLO

12 DE ENERO DE 2026



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Sistema Inteligente de Monitoreo Multicámara mediante Visión por Computador e Inteligencia Artificial

Nombre del candidato(a): Benjamín Ignacio Ramírez Barril

Carrera / Grado: Ingeniería Civil Telemática

Campus: Casa Central Valparaíso ; **Departamento:** Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Marcos Zúñiga Barraza, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 21/01/2026 ; Firma: _____

Estudiante o Candidato(a):

Fecha: 20/01/2026 ; Firma: _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Resumen

El presente trabajo de memoria se aborda el diseño y desarrollo de un sistema inteligente de monitoreo multicámara basado en visión por computador e inteligencia artificial, enfocado en la detección y análisis de la afluencia de personas en entornos públicos. El proyecto surge ante la necesidad de disponer de una herramienta tecnológica que permita a operadores y entidades de control gestionar múltiples cámaras IP de forma simultánea, obteniendo métricas precisas de flujo de personas y niveles de ocupación en tiempo real.

El sistema propuesto integra un backend desarrollado en Python utilizando el framework FastAPI, y un frontend implementado en React.js. En el lado del procesamiento, se emplea el modelo YOLOv8 para la detección de personas en las transmisiones provenientes de cámaras IP, soportando flujos en tiempo real mediante el uso de OpenCV. La arquitectura permite manejar múltiples cámaras en paralelo, realizar conteo de personas, visualizar resultados en dashboards y exportar los datos obtenidos en formato CSV para su posterior análisis.

La contribución personal de este trabajo se centra en el desarrollo de la interfaz web, la integración con el backend y la implementación de un sistema de procesamiento concurrente para soportar múltiples cámaras IP, asegurando una latencia mínima y un desempeño eficiente. Asimismo, se evalúan métricas asociadas al rendimiento del sistema, tales como latencia, throughput y variaciones según resolución de video.

Los resultados demuestran la viabilidad del sistema para operar en tiempo real con varias cámaras simultáneas, manteniendo una latencia controlada y una visualización estable, sentando las bases para futuras ampliaciones hacia un sistema de vigilancia inteligente integral y escalable.

Palabras clave: visión por computador, inteligencia artificial, YOLOv8, cámaras IP, FastAPI, React.js, monitoreo multicámara.

Abstract

This thesis presents the design and development of an intelligent multi-camera monitoring system based on computer vision and artificial intelligence, aimed at detecting and analyzing people flow in public environments. The project addresses the need for an efficient technological tool that enables operators and control entities to manage multiple IP cameras simultaneously while obtaining accurate real-time pedestrian flow metrics and occupancy levels.

The proposed system integrates a backend developed in Python using the FastAPI framework and a frontend implemented in React.js. The processing stage employs the YOLOv8 model for person detection in IP camera video streams, supporting real-time processing through OpenCV. The architecture enables handling multiple cameras concurrently, performing people counting, displaying results through dashboards, and exporting collected data in CSV format for further analysis.

The personal contribution of this work focuses on the development of the web interface, backend integration, and implementation of a concurrent processing system capable of supporting multiple IP cameras while ensuring minimal latency and efficient performance. Metrics such as latency, throughput, and resolution-based performance were evaluated to validate system reliability.

The obtained results demonstrate the feasibility of the proposed system to operate in real time with multiple simultaneous cameras, maintaining low latency and stable visualization, establishing a foundation for future expansions toward a fully scalable intelligent monitoring platform.

Keywords: computer vision, artificial intelligence, YOLOv8, IP cameras, FastAPI, React.js, multi-camera monitoring.

Agradecimientos

En primer lugar, deseo expresar mi más sincero agradecimiento a mi profesor guía, Marcos Zúñiga, por su constante apoyo, orientación y disposición a lo largo de todo el desarrollo de esta memoria. Su guía fue fundamental tanto en la definición del enfoque técnico del proyecto como en la mejora continua del informe, aportando observaciones críticas que permitieron fortalecer el trabajo y encauzarlo de manera rigurosa.

Agradezco profundamente a mi familia, en especial a mi madre, a mi padre y a mis abuelos, quienes han sido un pilar incondicional durante toda mi formación universitaria. Su apoyo constante, confianza y aliento en los momentos difíciles fueron esenciales para perseverar y creer en mis propias capacidades. Este logro es también reflejo del esfuerzo y el cariño que siempre me han entregado.

Asimismo, deseo dedicar unas palabras de agradecimiento a una persona muy especial que formó parte importante de mi vida durante este proceso. Su compañía, apoyo incondicional y cariño sincero fueron fundamentales en momentos de dificultad, estrés y exigencia académica. Siempre estuvo presente, atenta y dispuesta a priorizar mi bienestar incluso en circunstancias complejas, acompañándome de manera constante a lo largo de mi vida universitaria. El afecto recibido, tanto de su parte como de su familia, dejó una huella profunda y valiosa que permanece en mi corazón y que agradezco con profundo respeto y gratitud.

Un agradecimiento muy especial merece mi gato Ghoul, mi fiel compañero durante gran parte de mi vida universitaria y, hasta el día de hoy, una fuente invaluable de compañía emocional. Su presencia constante, su capacidad casi intuitiva de acompañarme en momentos de estrés, cansancio o tristeza, y su cercanía silenciosa mientras estudiaba o trabajaba frente al computador —siempre en el mismo rincón, sobre su mantita— fueron un apoyo fundamental durante este proceso. En los momentos de alegría, supo compartirla; y en los momentos difíciles, supo estar. Agradezco profundamente haber contado con él durante esta etapa tan importante de mi vida, y espero que continúe acompañándome en las que vendrán.

También quiero agradecer a mis amigos, tanto a aquellos que conocí durante la universidad como a quienes formé fuera de ella. A mis amigos universitarios, agradezco profundamente el apoyo académico, las jornadas de estudio compartidas y los momentos de distensión que hicieron más llevadero el proceso formativo, desde largas conversaciones hasta salidas espontáneas a comer completos donde la Nony o compartir fuera de clases. En particular, quiero agradecer a Gao, Chulopan y Javier, quienes no solo fueron amigos, sino también un grupo de estudio fundamental a lo largo de la carrera. Su apoyo constante, el trabajo colaborativo y las horas de estudio compartidas resultaron clave para superar múltiples asignaturas, especialmente durante los últimos años de formación. De manera especial, agradezco a Gao, con quien compartí el proceso de estudio de forma continua hasta el último año de la carrera. A mis amigos del voleibol, gracias por el compañerismo, la constancia y el crecimiento colectivo que hemos vivido como equipo y como personas, especialmente a mi amigo Benja que conocí en el voleibol, que si bien conozco hace poco, me ha acompañado en las buenas como en las malas, tanto en mi vida cotidiana, como en mi vida deportiva, muchas gracias.

Finalmente, agradezco al deporte del voleibol en sí mismo, que se convirtió en un espacio fundamental de equilibrio durante los años de estudio. Entrenar, competir y compartir en torno a este deporte me permitió liberar tensiones, enfrentar momentos difíciles y mantener la motivación incluso en etapas de alta exigencia académica, una herramienta clave para cuidar tanto mi bienestar físico como mental.

Índice general

1	Introducción	6
1.1	Contexto	6
1.2	Problema a Resolver	6
1.3	Acercamiento a la Solución	7
1.4	Hipótesis	7
1.5	Objetivos	8
1.5.1	Objetivo general:	8
1.5.2	Objetivos específicos:	8
1.6	Estructura de la Memoria	8
2	Estado del Arte	10
2.1	Soluciones existentes	10
2.1.1	Frigate (Network Video Recorder open-source)	10
2.1.2	Region-Based People Counting (Wicaksono et al., 2024)	11
2.1.3	Minimizing FPS Degradation in Multi-Streaming Processing with YOLO (COM-PSAC 2024)	12
2.1.4	Soluciones comerciales	13
2.1.5	Otros trabajos y soluciones	15
2.2	Herramientas y Métodos (Marco teórico)	15
2.2.1	Detección de objetos	16
2.2.2	Versiones de YOLO y elección práctica	16
2.2.3	Conceptos clave (IoU, mAP, F1, error de conteo)	17
2.2.4	Procesamiento y codecs (FFmpeg, RTSP, GStreamer)	17
2.2.5	Métricas de rendimiento para sistemas multicámara	18
2.3	Discusión	18
3	Desarrollo de la solución	21
3.1	Análisis de Requerimientos	21
3.1.1	Requerimientos Funcionales (RF)	21
3.1.2	Requerimientos No Funcionales (RNF)	22
3.1.3	Requerimientos de Interfaz (RI)	22
3.2	Diseño de la Solución	22
3.2.1	Diagrama de Contexto	23
3.2.2	Diagrama de Arquitectura del Sistema	24
3.2.3	Enumeración de Módulos	25

3.2.4	Matriz de Requerimientos y Módulos	25
3.2.5	Descripción de Módulos	25
3.2.6	Módulo CAP — Captura de video	26
3.2.7	Módulo PROC — Procesamiento de imágenes	26
3.2.8	Módulo GUI — Interfaz Gráfica	26
3.2.9	Módulo LOG — Registro de eventos	27
3.2.10	Módulo ALR — Alertas	27
3.2.11	Módulo REP — Reportes	27
3.2.12	Módulo USR — Usuarios	28
3.2.13	Foco de Módulos en la Memoria	28
3.3	Implementación	28
3.3.1	Objetivo Técnico de la Implementación	28
3.3.2	Arquitectura General Implementada	29
3.3.3	Implementación del Backend	30
3.3.4	Implementación del Frontend	32
3.3.5	Integración FastAPI–React	37
3.3.6	Sistema de Métricas y Alertas	37
3.3.7	Optimización del Rendimiento	39
3.3.8	Pruebas Unitarias	40
4	Validación	47
4.1	Hipótesis y Objetivos de Validación	47
4.2	Metodología de Validación	48
4.2.1	Métricas Instrumentadas	48
4.2.2	Restricción de Tiempo Real	49
4.2.3	Protocolo de Medición	49
4.2.4	Arquitectura de Medición	50
4.2.5	Escenarios Evaluados	51
4.3	Resultados	52
4.4	Discusión	60
5	Conclusiones y Trabajo Futuro	62
5.1	Conclusiones	62
5.2	Trabajo Futuro	63

Capítulo 1

Introducción

1.1. Contexto

El monitoreo automatizado de personas en espacios públicos ha adquirido una relevancia creciente en la gestión operativa, la seguridad y la planificación de infraestructuras de transporte. En particular, la red del **Metro de Santiago** constituye un entorno crítico: según el Instituto Nacional de Estadísticas (INE), en agosto de 2024 la empresa transportó aproximadamente **57,4 millones de pasajeros**, consolidándose como uno de los sistemas de transporte urbano más utilizados en América Latina[1] [2][3]. Este volumen masivo de usuarios genera desafíos operativos permanentes en materia de control de flujo, seguridad y gestión de aglomeraciones, especialmente durante horas punta.

En este contexto, disponer de herramientas tecnológicas que permitan realizar una observación automatizada y en tiempo real del flujo de personas se vuelve esencial para la toma de decisiones estratégicas. El Metro de Santiago, en su *Informe de Gestión 2024*, enfatiza la necesidad de fortalecer los mecanismos de supervisión y análisis de la demanda mediante soluciones digitales basadas en inteligencia artificial, orientadas a la optimización de la operación y la experiencia de los usuarios [4].

La incorporación de cámaras IP como sensores visuales, junto con técnicas de visión por computador y aprendizaje profundo, ha permitido la creación de sistemas capaces de detectar y contabilizar personas automáticamente. Sin embargo, estos sistemas enfrentan importantes desafíos cuando se extienden a entornos multicámara. Trabajos recientes demuestran que el procesamiento simultáneo de múltiples flujos de video incrementa la latencia y la carga computacional de forma no lineal, comprometiendo la capacidad de respuesta del sistema en escenarios en tiempo real [6] [7].

Por lo tanto, surge la necesidad de desarrollar una solución escalable y eficiente que permita monitorear varias cámaras IP de manera concurrente, manteniendo una latencia controlada y una calidad de detección adecuada para entornos operativos exigentes como el Metro de Santiago.

1.2. Problema a Resolver

Los sistemas tradicionales de videovigilancia presentan una serie de limitaciones cuando se trata de gestionar múltiples cámaras IP de forma simultánea. En la mayoría de los casos, estos sistemas fueron diseñados para operar en arquitecturas centralizadas o monolíticas, lo que implica una dependencia directa del hardware disponible y una limitada capacidad de escalamiento [8]. Esto conlleva problemas de latencia, degradación del rendimiento y dificultades en la integración de nuevos flujos de video.

En entornos de alta densidad de tránsito, como las estaciones de Metro, estas deficiencias se vuelven críticas. Un retraso de segundos en la detección de aglomeraciones puede afectar la toma de decisiones, la seguridad de los usuarios o la planificación del flujo de pasajeros. Según datos operacionales de Metro S.A., los horarios punta concentran hasta un 40 % del flujo diario total de pasajeros, lo que evidencia la necesidad de contar con soluciones que permitan cuantificar y monitorear el comportamiento de la afluencia en tiempo real [4].

Trabajos recientes, como los de Wicaksono et al. (2024) y Chung et al. (2024), demuestran que el manejo simultáneo de múltiples flujos genera una carga no lineal sobre los recursos computacionales, afectando la latencia, la tasa de cuadros por segundo (FPS) y la estabilidad general del sistema [6] [7]. Esto se traduce en una pérdida de precisión en la detección y una menor capacidad de respuesta ante cambios bruscos en la densidad de personas.

En consecuencia, el problema a resolver en esta memoria consiste en diseñar e implementar un sistema capaz de **procesar múltiples cámaras IP en paralelo** con detección y conteo de personas en tiempo real, asegurando un desempeño estable, una baja latencia y la posibilidad de escalar según los requerimientos del entorno..

1.3. Acercamiento a la Solución

La solución propuesta consiste en el desarrollo de un sistema distribuido basado en una arquitectura cliente-servidor, donde el backend —implementado en **FastAPI (Python)**— se encarga del procesamiento de los flujos de video, y el frontend —desarrollado en **React.js**— proporciona una interfaz moderna, responsiva y centralizada para el operador. Este enfoque permite una administración eficiente de múltiples cámaras, la visualización en tiempo real y la generación de métricas automáticas de afluencia.

Cada cámara IP se conecta mediante protocolos estándar como RTSP o MJPEG, y los flujos se procesan en paralelo utilizando hilos independientes, lo que permite mantener la eficiencia incluso en escenarios multicámara. Para la detección de personas se emplea el modelo **YOLOv8**, reconocido por su balance entre velocidad de inferencia y precisión, siendo ampliamente adoptado en aplicaciones de vigilancia inteligente y monitoreo de multitudes [9] [14] .

La arquitectura también integra mecanismos de exportación de datos en formato CSV, dashboards visuales y gráficos históricos, permitiendo que los operadores obtengan una visión analítica del comportamiento de la afluencia a lo largo del tiempo. De esta forma, el sistema se adapta tanto a las necesidades operativas del Metro de Santiago como a los requerimientos técnicos de un entorno de procesamiento intensivo.

1.4. Hipótesis

Es posible implementar un sistema inteligente de monitoreo multicámara capaz de detectar y contabilizar personas en tiempo real, manteniendo una latencia cliente-servidor inferior a 500ms y un rendimiento estable con al menos **cinco cámaras IP simultáneas** en un equipo de especificaciones estándar.

Dicha hipótesis será validada experimentalmente en la etapa de pruebas, mediante la medición de métricas de **latencia, throughput y uso de recursos computacionales**, en distintos escenarios

de resolución y número de flujos activos, con el objetivo de determinar la escalabilidad práctica del sistema.

1.5. Objetivos

1.5.1. Objetivo general:

Desarrollar un sistema de monitoreo inteligente de afluencia de personas en el Metro de Santiago mediante el uso de cámaras y algoritmos de visión por computador, que permita entregar información en tiempo real sobre el flujo de pasajeros, facilitando la toma de decisiones operativas y de seguridad. El sistema deberá ser escalable para su aplicación en otros entornos públicos con alta concentración de personas.

1.5.2. Objetivos específicos:

- OE1: Integrar y recopilar fuentes de video provenientes de cámaras públicas y entornos reales, como estaciones del Metro de Santiago, para alimentar y validar un sistema de detección y conteo de personas en tiempo real basado en modelo de IA preentrenado.
- OE2: Implementar y optimizar el procesamiento de video en tiempo real utilizando un modelo preentrenado de detección de personas, aplicando ajustes de parámetros y técnicas de seguimiento para mejorar la estabilidad del tracking y la precisión del conteo.
- OE3: Implementar una arquitectura de software escalable que soporte la gestión simultánea de múltiples cámaras IP mediante procesamiento concurrente.
- OE4: Diseñar una interfaz web interactiva que permita visualizar en tiempo real las detecciones, métricas de afluencia y estadísticas históricas.
- OE5: Evaluar el desempeño del sistema en términos de latencia, throughput y uso de recursos, considerando distintas resoluciones y configuraciones de cámaras.

En el marco del desarrollo de esta memoria, el foco personal se centra en la **implementación de la arquitectura multicámara (OE3)**, la integración entre el backend y el frontend (OE4), y la evaluación del rendimiento del sistema bajo condiciones de operación reales, con especial atención a la eficiencia y estabilidad del procesamiento simultáneo (OE5).

1.6. Estructura de la Memoria

El presente documento se estructura de la siguiente forma:

- **Capítulo 2: Estado del Arte.** Revisa soluciones existentes, herramientas, marco teórico y contribución con la propuesta desarrollada.
- **Capítulo 3: Desarrollo de la Solución.** Detalla el diseño, implementación y pruebas unitarias del sistema propuesto, con énfasis en la arquitectura multicámara.
- **Capítulo 4: Validación.** Presenta las métricas y resultados experimentales obtenidos, junto con el análisis del desempeño en distintos escenarios.

- **Capítulo 5: Conclusiones y Trabajo Futuro.** Resume los logros alcanzados, limitaciones encontradas y posibles líneas de mejora para el sistema.

Capítulo 2

Estado del Arte

2.1. Soluciones existentes

En esta sección se presentan y discuten las soluciones, proyectos y estudios que más se aproximan al problema abordado en esta memoria: sistemas de monitoreo de afluencia de personas basados en cámaras IP y modelos de visión por computador. Se incluyen tanto productos/servicios disponibles en el mercado, como proyectos open-source relevantes (por su arquitectura multicámara o por su enfoque en endpoint/ edge) y trabajos académicos recientes que informan sobre aspectos críticos de rendimiento (métricas de latencia, throughput y técnicas para manejar múltiples flujos).

2.1.1. Frigate (Network Video Recorder open-source)

Frigate es un NVR de código abierto orientado a la detección de objetos en tiempo real en cámaras IP. Proporciona procesamiento local (edge) y está pensado para integrarse con plataformas domóticas o de monitoreo; en su diseño prioriza la ejecución local de modelos y la posibilidad de aprovechar aceleradores (por ejemplo Coral/TPU o técnicas de optimización como OpenVINO). Frigate se usa a menudo como referencia práctica cuando se quiere una solución local, basada en cámaras IP y con salida a una interfaz web para visualización y alertas, lo cual lo hace relevante en el contexto de soluciones prácticas que no requieren un servicio en la nube centralizado [\[5\]](#).

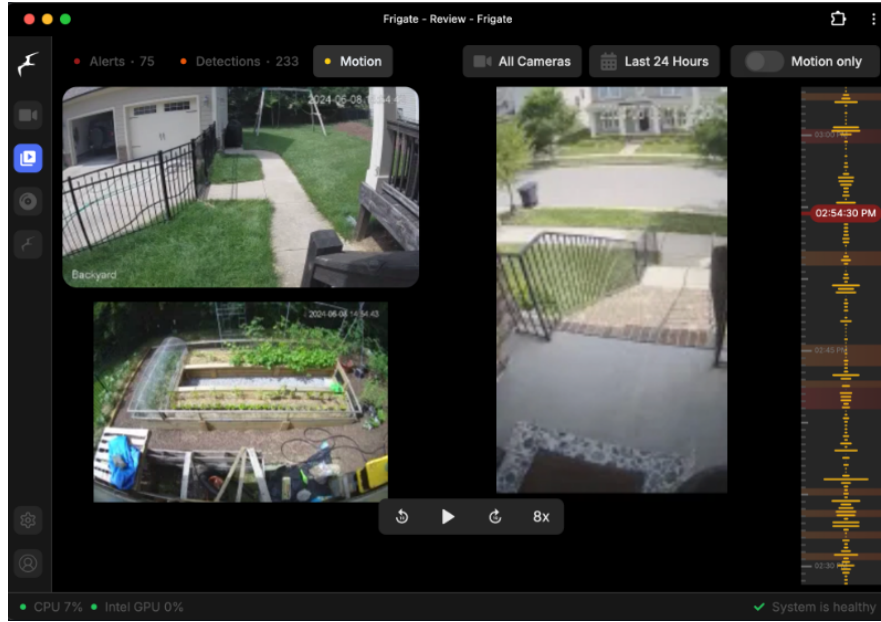


Figura 2.1: Multi-camera scrubbing Frigate.

2.1.2. Region-Based People Counting (Wicaksono et al., 2024)

Wicaksono et al.[6] presentan una implementación práctica orientada a *smart buildings* que combina YOLOv4-tiny (detección) y KLT (tracking) en un sistema embebido acelerado con NVIDIA DeepStream. El estudio es relevante por su enfoque multicámara RTSP → dispositivo embebido → salida RTSP, y por reportar medidas concretas de desempeño: para un solo stream se obtienen ~23.2 FPS; con dos streams la salida se ubica en el rango 9.3–10.7 FPS y la latencia de entrada/salida puede situarse entre 1–2 s en condiciones de prueba. El trabajo además discute límites prácticos (p. ej. degradación de latencia de salida al aumentar el framerate de entrada), por lo que es útil como comparador práctico para sistemas embebidos multicámara.

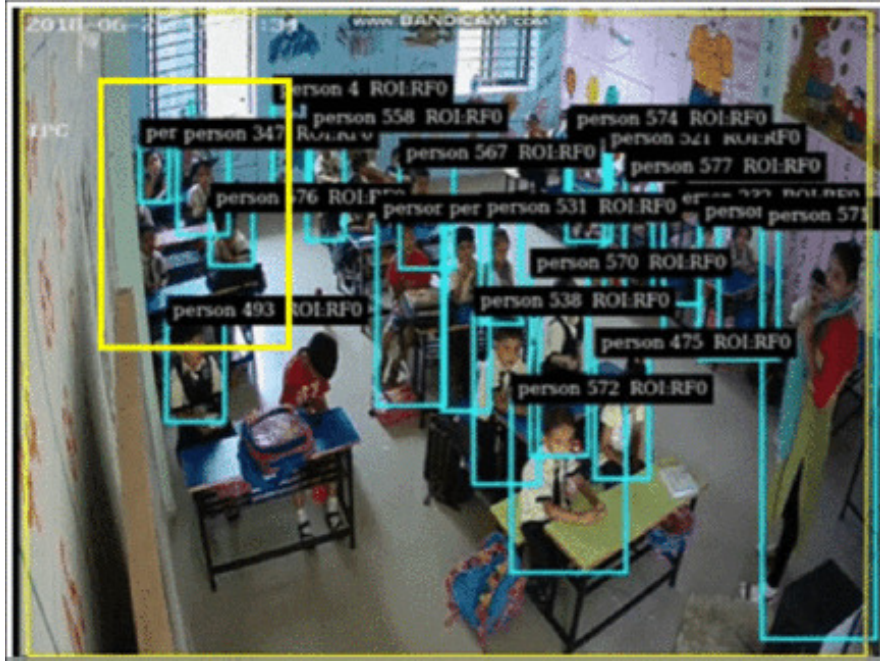


Figura 2.2: Vídeo de prueba para situaciones con mucha gente con el detector YOLOv4 y KLT.

2.1.3. Minimizing FPS Degradation in Multi-Streaming Processing with YOLO (COMPSAC 2024)

Este trabajo aborda directamente el problema que ocupa a esta memoria: cómo conservar FPS y evitar la degradación cuando múltiples flujos requieren inferencia simultánea con YOLO. La aportación clave es una estrategia de *object-presence detection* liviana (MSE / RGA) que activa la inferencia pesada sólo cuando existe movimiento/objeto; cuando aparecen objetos, se graban segmentos y se procesan en cola secuencial para evitar que YOLO se ejecute simultáneamente en todas las cámaras. Experimentalmente, el sistema mantiene FPS estables hasta 8 cámaras y acepta un retardo medio en cola (≈ 4 s), planteando una alternativa práctica al enfoque *YOLO-only* para escenarios con recursos limitados [7].

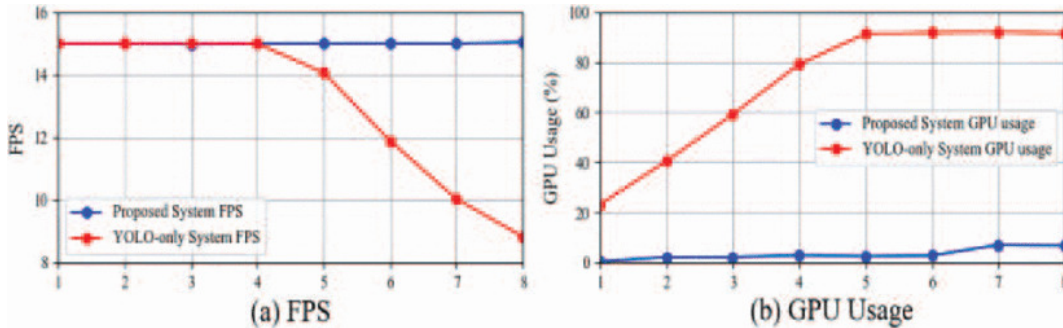


Figura 2.3: Resultado del uso de FPS y GPU cambiando según el número de transmisiones.

2.1.4. Soluciones comerciales

Axis Camera Station

Axis Camera Station es un sistema de gestión de video (VMS) desarrollado por Axis Communications, orientado a aplicaciones de videovigilancia y análisis de comportamiento en entornos corporativos e institucionales. Este software permite la administración centralizada de múltiples cámaras IP, incluyendo grabación, detección de movimiento, análisis de ocupación y generación de alertas automatizadas. Su arquitectura cliente-servidor se basa en la comunicación RTSP y HTTP(S), permitiendo integrar dispositivos Axis y de terceros compatibles con ONVIF. Entre sus funcionalidades destacan la visualización en tiempo real de flujos simultáneos, la generación de reportes de actividad y el control remoto de cámaras PTZ.

A nivel técnico, Axis Camera Station ofrece detección y conteo de personas mediante analítica integrada optimizada por hardware especializado, lo que lo posiciona como un referente comercial en soluciones multicámara de escala industrial. No obstante, su naturaleza propietaria, su dependencia de licencias y la falta de acceso al procesamiento interno limitan su flexibilidad para investigaciones o adaptaciones personalizadas. En este sentido, la solución desarrollada en esta memoria se diferencia por ofrecer una arquitectura abierta, de código modificable, y basada en tecnologías libres (*FastAPI*, *React.js*, *OpenCV*), permitiendo adaptar los algoritmos y parámetros de detección según las necesidades de cada escenario [10].

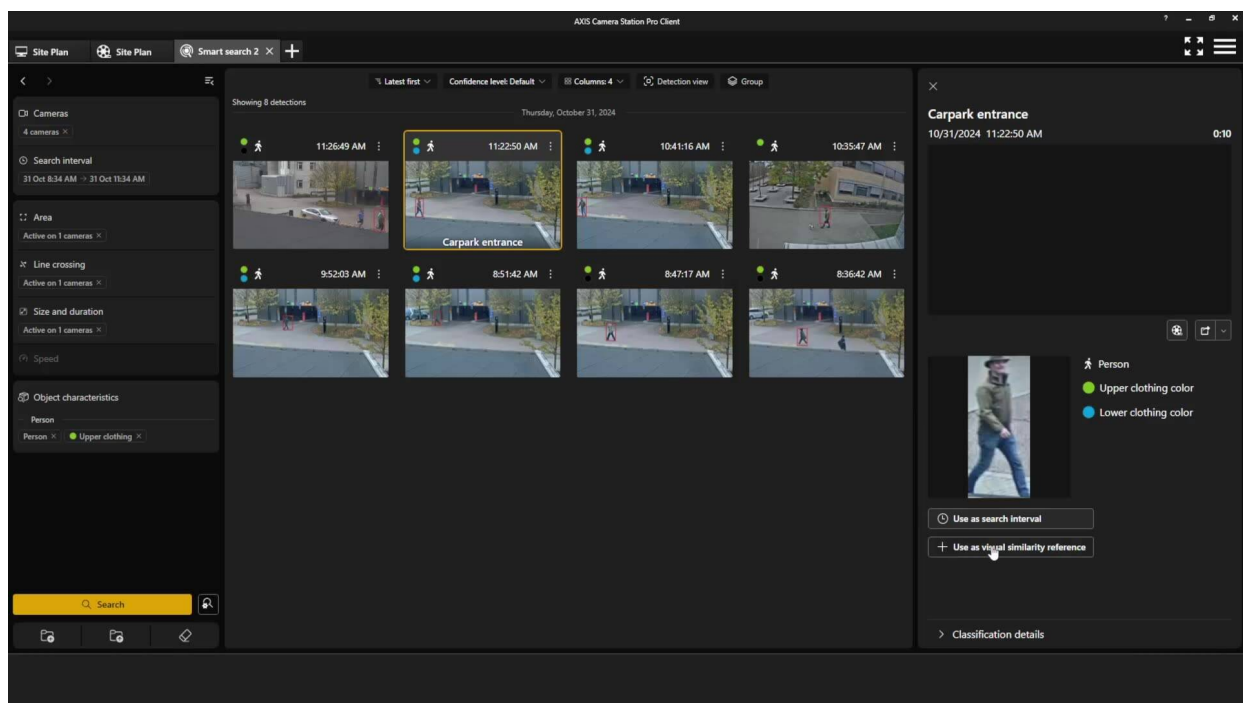


Figura 2.4: Interfaz de Axis Camera Station Pro.

Milestone XProtect

Milestone XProtect es una plataforma de gestión de video (VMS) desarrollada por Milestone Systems, ampliamente utilizada en entornos empresariales, gubernamentales y de transporte público. Este sistema permite la administración, grabación y monitoreo de cientos de cámaras IP de forma

simultánea, ofreciendo una arquitectura distribuida basada en servidores y clientes que facilita la escalabilidad en implementaciones multicámara de gran tamaño. XProtect integra compatibilidad con más de 10.000 dispositivos a través del estándar ONVIF y protocolos RTSP/HTTP, asegurando interoperabilidad con diversas marcas de cámaras y dispositivos IoT.

Entre sus funcionalidades destacadas se encuentran el análisis de video en tiempo real, la detección de movimiento, la gestión avanzada de usuarios y la integración con módulos de analítica de terceros. Además, XProtect proporciona herramientas de búsqueda inteligente, mapas interactivos y reproducción sincronizada de múltiples cámaras, características que lo consolidan como una solución de referencia en videovigilancia profesional. Sin embargo, su naturaleza cerrada y el alto costo asociado a licencias y mantenimiento limitan su adopción en entornos académicos o proyectos experimentales. A diferencia de la solución desarrollada en esta memoria, XProtect no ofrece acceso al pipeline interno de procesamiento, lo que restringe la posibilidad de experimentar con métricas de rendimiento, ajuste de modelos de detección o personalización de la inferencia multicámara. [11]

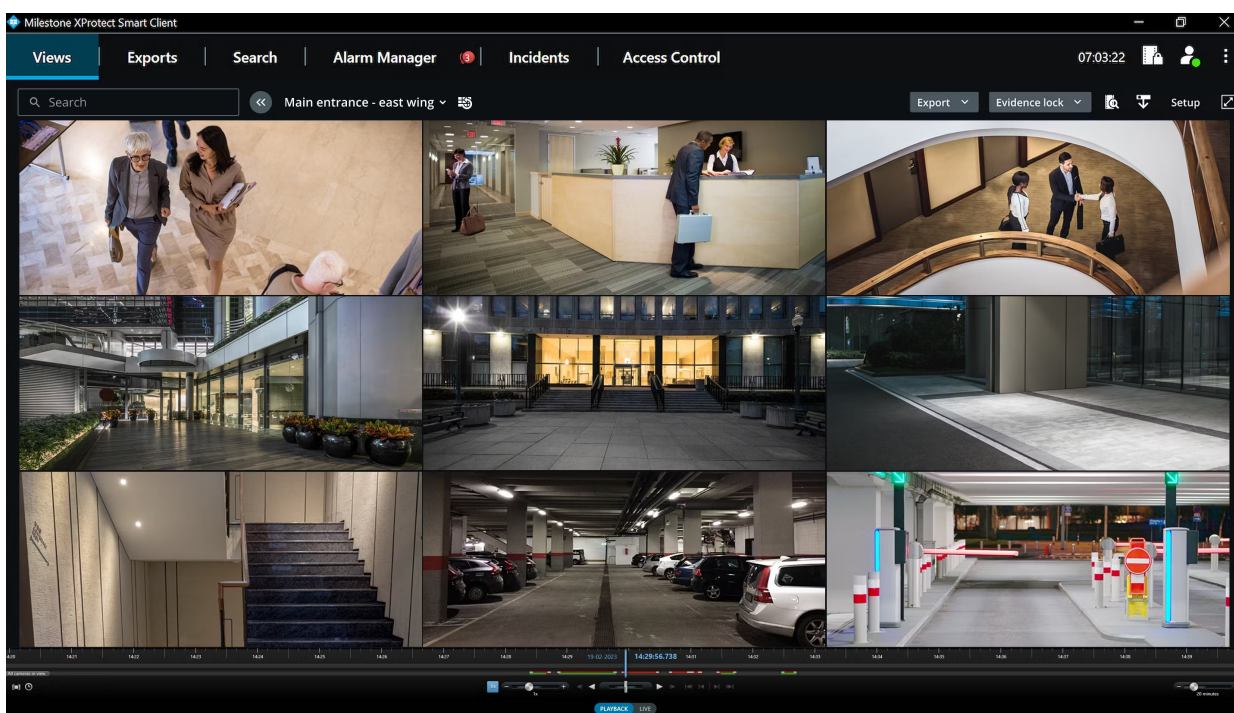


Figura 2.5: Interfaz de Milestone XProtect Smart Client .

Avigilon Control Center

Avigilon Control Center (ACC), desarrollado por Motorola Solutions, destacada en el ámbito de la videovigilancia inteligente. Está orientado a instalaciones como aeropuertos, sistemas de transporte masivo y campus universitarios, ofreciendo una gestión centralizada de video con analítica avanzada basada en inteligencia artificial. Su arquitectura cliente-servidor permite visualizar, grabar y analizar flujos provenientes de múltiples cámaras IP con una latencia reducida, manteniendo una actualización fluida incluso en entornos con decenas de transmisiones concurrentes.

ACC incorpora algoritmos de reconocimiento facial, detección de objetos y análisis de movimiento optimizados mediante hardware dedicado (GPU/TPU), lo que mejora su desempeño en escenarios de alta demanda. Asimismo, proporciona una interfaz unificada que integra mapas interactivos, alertas

automáticas y generación de reportes, posibilitando una supervisión integral desde un solo panel de control. Pese a su robustez técnica, su carácter propietario y los requerimientos de hardware certificados reducen su flexibilidad para entornos de investigación o desarrollos experimentales. En contraposición, el sistema propuesto en esta memoria prioriza la apertura, la modularidad y la posibilidad de realizar pruebas controladas sobre la arquitectura de transmisión y procesamiento, sin equipamiento específicos. [12][13]

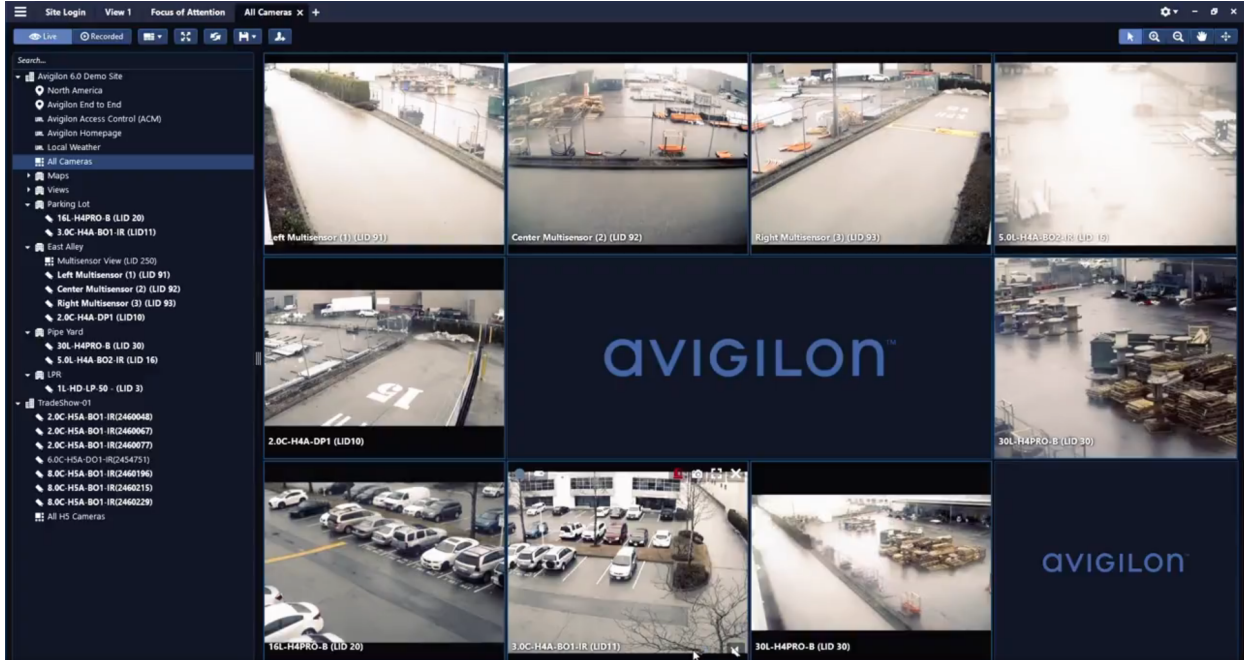


Figura 2.6: Interfaz de Avigilon Control Center.

2.1.5. Otros trabajos y soluciones

Además de los ejemplos anteriores, existen estudios y herramientas que aportan evidencias y técnicas aplicables al problema:

- Estudios sobre conteo y análisis en estaciones/tránsito público (varios trabajos en *Sensors* y conferencias) que muestran pipelines RTSP + detector + ROI para conteos y heatmaps. Estas referencias aportan metodología para datasets, pruebas y métricas de desempeño. [8]
- Preprints recientes que aportan mejoras en arquitecturas de detección y/o pipelines multicámara y optimizaciones a nivel de inferencia (ej.: arXiv:2401.14132). [9]
- SDKs comerciales/industrial-grade para multicámara y edge: NVIDIA DeepStream (GStreamer + pipelines aceleradas), Intel OpenVINO (optimización de modelos para CPU/NPU) — ambos usados con frecuencia para producir NVRs o sistemas embebidos de alto rendimiento.[19]

2.2. Herramientas y Métodos (Marco teórico)

En este apartado se describen las principales herramientas, técnicas y fundamentos teóricos que sustentan el desarrollo de un sistema inteligente de monitoreo de afluencia de personas basado en visión por computador e inteligencia artificial. El propósito de este marco es contextualizar las decisiones

tecnológicas adoptadas durante la implementación, las cuales buscan lograr un equilibrio entre precisión de detección, eficiencia computacional y escalabilidad en escenarios donde se manejan múltiples cámaras IP de manera simultánea. La revisión de los principios y algoritmos involucrados permite comprender los componentes esenciales que hacen posible la detección y conteo de personas en tiempo real, así como la forma en que estos se integran para dar soporte a la arquitectura propuesta.

2.2.1. Detección de objetos

La detección de objetos constituye uno de los pilares fundamentales en la visión por computador moderna. Este proceso consiste en identificar y localizar instancias de objetos dentro de una imagen o flujo de video, asignándoles una etiqueta de clase y una región delimitadora. Comúnmente, se utilizan redes neuronales convolucionales (CNN), que han demostrado una notable capacidad para extraer características visuales complejas y representar patrones espaciales en distintas escalas. Existen dos enfoques dominantes: los detectores de dos etapas, como R-CNN, Faster R-CNN y Mask R-CNN, que priorizan la precisión a costa de un mayor tiempo de procesamiento, y los detectores de una etapa, como SSD y YOLO, que simplifican el pipeline de inferencia, permitiendo resultados en tiempo real [14].

Dado que el sistema desarrollado en esta memoria tiene como objetivo principal procesar flujos de video en tiempo real provenientes de múltiples cámaras IP, la elección de un detector de una sola etapa resulta natural. Este tipo de modelo, al integrar las etapas de localización y clasificación en una única pasada, reduce la latencia y facilita el procesamiento paralelo. En particular, la familia de modelos YOLO (*You Only Look Once*) ha demostrado un excelente balance entre velocidad y precisión, siendo adoptada ampliamente en aplicaciones donde la respuesta inmediata es esencial [15].

2.2.2. Versiones de YOLO y elección práctica

El modelo YOLO ha pasado por diversas etapas de evolución, cada una introduciendo mejoras tanto en la arquitectura como en la velocidad de inferencia. Desde su primera versión, YOLOv1, hasta las más recientes como YOLOv8, se han incorporado nuevas estrategias de aprendizaje, bloques convolucionales optimizados y técnicas de regularización que han mejorado significativamente su rendimiento en entornos complejos [14]. Las versiones intermedias, como YOLOv3 y YOLOv4, marcaron hitos al incluir capas residuales y un mejor manejo de escalas, mientras que YOLOv5 y YOLOv7 introdujeron variaciones más ligeras pensadas para la ejecución en dispositivos embebidos o entornos de baja potencia.

La versión YOLOv8, desarrollada por Ultralytics [15], representa una mejora sustancial en términos de flexibilidad y eficiencia. Su diseño modular, junto con la posibilidad de escoger entre distintas variantes según el tamaño del modelo (n, s, m, l, x), permite adaptarse a distintas necesidades de rendimiento. En el contexto de esta memoria, se seleccionó la versión YOLOv8s, que ofrece un punto medio entre velocidad y precisión, siendo adecuada para un sistema que debe procesar múltiples flujos simultáneamente sin incurrir en sobrecarga computacional. Además, este modelo incorpora mejoras derivadas del paradigma *anchor-free*, reduciendo la cantidad de parámetros y acelerando el proceso de entrenamiento e inferencia. En sistemas multicámara, donde la carga de trabajo crece proporcionalmente al número de flujos activos, disponer de un modelo liviano es un requisito esencial para mantener un rendimiento estable.

El modelo seleccionado es capaz de detectar la clase “persona” con una precisión competitiva, incluso en entornos de iluminación variable o con oclusiones parciales, características típicas de espacios públicos. Al mismo tiempo, su integración con el framework de procesamiento permite aprovechar el paralelismo de GPU y técnicas como la cuantización o el uso de operaciones FP16, optimizando así el tiempo de inferencia. De este modo, la elección de YOLOv8s responde tanto a criterios de precisión como de escalabilidad, condición indispensable para validar el desempeño del sistema bajo condiciones de alta concurrencia.

2.2.3. Conceptos clave (IoU, mAP, F1, error de conteo)

La evaluación del desempeño de un modelo de detección se realiza mediante un conjunto de métricas estandarizadas que permiten cuantificar tanto la exactitud de las predicciones como su consistencia en diferentes escenarios. Una de las métricas más utilizadas es el **IoU (Intersection over Union)**, que mide el grado de solapamiento entre la caja predicha por el modelo y la región de referencia (*ground truth*). Un valor de IoU alto indica que la detección se aproxima con precisión a la ubicación real del objeto. Sobre la base del IoU, se calculan indicadores derivados como la precisión, el recall, el F1-score y el **mAP (mean Average Precision)**, que resumen el comportamiento global del detector frente a distintos umbrales de confianza.

En aplicaciones como el conteo de personas, la exactitud de la detección no siempre es el único parámetro de interés. También se evalúa el **error de conteo**, que representa la diferencia porcentual entre el número de personas detectadas y el conteo real observado. En contextos operativos, especialmente cuando el sistema se utiliza para estimar afluencia o congestión, es más relevante mantener un conteo fluido y estable que lograr la detección perfecta de cada individuo. Estas métricas son empleadas tanto en las pruebas experimentales del sistema desarrollado como en la validación de las soluciones revisadas, sirviendo de base para comparar su desempeño bajo distintas condiciones.

2.2.4. Procesamiento y codecs (FFmpeg, RTSP, GStreamer)

El procesamiento de video en tiempo real requiere una infraestructura capaz de recibir, decodificar y manipular flujos provenientes de distintas cámaras, con formatos y protocolos variables. La mayoría de las cámaras IP utilizan el protocolo RTSP (Real Time Streaming Protocol) o flujos en MJPEG/HTTP para la transmisión de video. Herramientas como FFmpeg [18] y OpenCV ofrecen funciones para la captura y decodificación de estos flujos, permitiendo obtener cada cuadro como una matriz de píxeles apta para el procesamiento por los modelos de detección. En escenarios de mayor complejidad, donde se requiere optimizar el rendimiento y aprovechar recursos de hardware especializados, frameworks como GStreamer y DeepStream [19] resultan particularmente útiles, ya que permiten diseñar pipelines paralelizados y aprovechar aceleradores como GPUs o NPUs.

El sistema desarrollado en esta memoria emplea OpenCV para la decodificación de los flujos y la gestión del procesamiento de imágenes, integrándolo con el backend implementado en FastAPI. Esta combinación ofrece un entorno flexible que permite manejar múltiples flujos en paralelo, enviar los resultados procesados a la interfaz gráfica y mantener una latencia baja en la comunicación. A diferencia de las soluciones tradicionales que procesan los videos de manera secuencial, la arquitectura propuesta utiliza hilos independientes para cada cámara, lo que permite escalar el sistema según la capacidad de hardware disponible. Este enfoque asegura una operación estable incluso al aumentar la cantidad de cámaras, lo que resulta fundamental para validar el desempeño en entornos reales.

2.2.5. Métricas de rendimiento para sistemas multicámara

En sistemas de monitoreo multicámara, las métricas de rendimiento adquieren un papel crucial, ya que permiten cuantificar la capacidad del sistema para procesar información de manera eficiente y sin degradar la calidad del servicio. Entre las métricas más relevantes se encuentran la latencia cliente-servidor, el throughput y la escalabilidad. La **latencia cliente-servidor** representa el tiempo total que transcurre desde que un cuadro es capturado por la cámara hasta que su resultado es mostrado en la interfaz o almacenado en la base de datos. Este parámetro incluye la decodificación del flujo, la inferencia del modelo, la transmisión de datos y la actualización visual, y es fundamental para determinar si el sistema puede considerarse realmente en tiempo real. El **throughput**, por su parte, mide la cantidad de cuadros procesados por segundo (FPS), tanto por cámara individual como a nivel del sistema global. Una disminución significativa en este valor puede afectar la fluidez de la visualización y comprometer la detección de eventos dinámicos.

El **uso de recursos computacionales**, que incluye CPU, GPU y memoria, también constituye un indicador clave, especialmente cuando se manejan varios flujos concurrentes. Trabajos recientes, como los de Wicaksono et al. [6] y Chung et al. [7], han demostrado que el procesamiento simultáneo de múltiples cámaras genera una carga no lineal sobre los recursos, afectando la latencia y la tasa de cuadros. En este sentido, el sistema propuesto busca evaluar y optimizar estos parámetros, garantizando un desempeño estable mediante la ejecución asíncrona de procesos y la posibilidad de aplicar estrategias de reducción de carga. Finalmente, la **escalabilidad** se define como la capacidad del sistema para mantener un rendimiento aceptable al aumentar el número de cámaras activas o la resolución de los flujos. Medir esta propiedad es esencial para validar la hipótesis principal de esta memoria, que plantea la viabilidad de un sistema multicámara en tiempo real sin sacrificar precisión ni estabilidad.

En conjunto, los fundamentos técnicos y teóricos descritos en este marco establecen las bases que sustentan el desarrollo del sistema propuesto. La combinación de detección eficiente, procesamiento concurrente y arquitectura modular permite abordar los desafíos asociados a la operación con múltiples cámaras, sentando las condiciones necesarias para la validación experimental que se realizará en los capítulos posteriores.

2.3. Discusión

A partir del análisis de las soluciones existentes, es posible observar que el área de monitoreo de afluencia de personas mediante visión por computador ha evolucionado hacia modelos cada vez más eficientes y escalables, especialmente con la aparición de sistemas multicámara que permiten ampliar la cobertura espacial y mejorar la precisión del conteo. Sin embargo, la mayoría de los trabajos revisados se enfocan en escenarios de laboratorio o en entornos controlados, donde las condiciones de iluminación, el posicionamiento de las cámaras y el número de flujos simultáneos están restringidos. La propuesta desarrollada en esta memoria busca superar estas limitaciones mediante el diseño de un sistema robusto, capaz de operar en entornos reales con múltiples cámaras IP, integrando técnicas de detección y procesamiento eficientes, y manteniendo un control explícito sobre los recursos computacionales disponibles.

En comparación con herramientas open-source como Frigate, que ofrece detección de objetos en tiempo real para aplicaciones domésticas o de seguridad, la solución propuesta se distingue por su orientación hacia la gestión centralizada de múltiples cámaras en entornos públicos, priorizando la

escalabilidad y la administración dinámica de flujos. Mientras *Frigate* depende de una arquitectura monolítica enfocada en el procesamiento local y en el uso de aceleradores de hardware, el sistema desarrollado en esta memoria se estructura bajo una arquitectura modular basada en microservicios, utilizando **FastAPI** para el backend y **React.js** para la interfaz gráfica. Esta separación de responsabilidades podrá permitir un mantenimiento más flexible, facilitará la integración con nuevas fuentes de video y posibilita la futura incorporación de una capa de persistencia en base de datos para el almacenamiento histórico de métricas.

Del mismo modo, los trabajos académicos revisados, como el de Wicaksono et al. (2024) y el presentado en COMPSAC (2024), abordan problemáticas muy específicas de la inferencia en tiempo real, como la degradación de FPS o la latencia al procesar múltiples flujos simultáneamente. Aunque ambos estudios entregan resultados valiosos y metodologías optimizadas para entornos embebidos o de baja capacidad, ninguno de ellos plantea una plataforma integral que combine la gestión visual, la visualización analítica y la operación multicámara orientada a un operador humano. En contraste, la solución aquí propuesta no solo considera el procesamiento distribuido de cámaras IP, sino también la presentación de información analítica a través de dashboards interactivos que permiten interpretar el comportamiento de la afluencia de personas en tiempo real, aportando un valor operativo directo al usuario final.

En el ámbito de las soluciones comerciales, herramientas consolidadas como **Axis Camera Station**, **Milestone XProtect** y **Avigilon Control Center** representan el estado del arte en la gestión de sistemas multicámara a gran escala. Estas plataformas destacan por su estabilidad, su capacidad de procesamiento distribuido y sus interfaces de operación profesional, permitiendo la administración centralizada de múltiples transmisiones simultáneas con bajo retardo. No obstante, su naturaleza propietaria y el enfoque cerrado de sus arquitecturas limitan la posibilidad de experimentar con los parámetros internos de inferencia, ajustar dinámicamente los umbrales de detección o analizar la degradación del rendimiento a medida que aumenta el número de flujos. En contraste, el sistema propuesto en esta memoria plantea una alternativa abierta y configurable, donde la integración del procesamiento, la interfaz y la gestión se realiza de manera transparente, favoreciendo la investigación y optimización de métricas como la latencia, el throughput y la eficiencia de escalamiento.

La arquitectura propuesta se fundamenta en los principios descritos en el marco teórico. El módulo de detección de personas utiliza el modelo **YOLOv8s**, seleccionado por su equilibrio entre velocidad de inferencia y precisión, lo que lo hace adecuado para entornos donde la latencia es un parámetro crítico. La detección se complementa con un sistema de conteo basado en el seguimiento temporal de las detecciones por cuadro, lo que permite evitar duplicaciones y obtener métricas consistentes de flujo. Cada cámara IP es procesada de manera independiente por un hilo dedicado, empleando el módulo de procesamiento de video (*video_processor.py*), lo que posibilita una ejecución concurrente que se adapta al número de cámaras activas. Este diseño difiere sustancialmente de las soluciones revisadas, que tienden a emplear un pipeline único y secuencial, generando cuellos de botella a medida que aumenta el número de flujos simultáneos.

Otro aspecto diferenciador de la propuesta es la integración entre el procesamiento en tiempo real y la capa de presentación. Mediante **WebSockets** y **FastAPI Streaming**, se transmite el video procesado hacia el frontend en formato MJPEG, asegurando una comunicación continua y estable con baja latencia. Este enfoque contrasta con las soluciones tradicionales que dependen de almacenamiento intermedio o de la actualización por intervalos, las cuales añaden retardo perceptible para el operador. Además, el sistema permite registrar las métricas de conteo en estructuras temporales en memoria y

exportarlas en formato CSV, garantizando la trazabilidad de los datos aun cuando la conexión del operador no esté activa, lo que responde directamente a la necesidad de persistencia offline planteada en los requerimientos del proyecto.

El componente de interfaz gráfica constituye también un aporte significativo en relación con el estado del arte. La aplicación desarrollada en React.js proporciona una experiencia moderna y minimalista que facilita la administración de múltiples cámaras, permitiendo añadir, eliminar o visualizar flujos en tiempo real con información complementaria como nombre, ubicación geográfica y dirección IP. Esta interfaz, además de ofrecer un control intuitivo, presenta dashboards dinámicos que muestran la evolución del conteo de personas a lo largo del tiempo, combinando el procesamiento visual con un análisis de datos básico, algo que las soluciones revisadas no contemplan de forma integrada. De esta manera, la propuesta combina elementos de detección, gestión y visualización en un solo sistema coherente y extensible.

Desde una perspectiva de eficiencia y validación, la solución propuesta busca abordar de manera experimental los desafíos mencionados en los estudios existentes: medir de forma sistemática la latencia cliente-servidor, el throughput por cámara y el uso de recursos computacionales a distintas resoluciones. La evaluación empírica del sistema permitirá determinar los límites prácticos de procesamiento concurrente, replicando y extendiendo los análisis de degradación de FPS presentados en trabajos previos, pero aplicados a un entorno de arquitectura web modular y escalable. En este sentido, la memoria no se limita a reproducir resultados académicos, sino que los traslada al contexto de un sistema operacional real con potencial de implementación en infraestructuras de monitoreo público o institucional.

En conclusión, la propuesta de esta memoria se diferencia de las soluciones revisadas tanto en su enfoque arquitectónico como en su propósito operativo. Mientras la mayoría de las investigaciones se centran en optimizar el rendimiento de la detección a nivel algorítmico o en dispositivos específicos, esta memoria integra los componentes de procesamiento, visualización y gestión en una solución completa, validando su desempeño a través de métricas de latencia, throughput y escalabilidad. La integración de múltiples cámaras IP en un entorno unificado, la persistencia de datos aun en modo offline, y la estructura modular orientada a la expansión futura, constituyen las principales contribuciones técnicas del trabajo. Estos elementos, en conjunto, permiten considerar el sistema propuesto como una plataforma adaptable y viable para la supervisión inteligente de afluencia de personas en entornos reales.

Capítulo 3

Desarrollo de la solución

El desarrollo de la solución se realiza bajo una metodología de trabajo iterativa e incremental, organizada en hitos que abarcaron desde el análisis inicial de requerimientos hasta la validación funcional de los módulos desarrollados. Cada etapa permite consolidar la arquitectura modular del sistema y definir las interacciones entre sus componentes principales, garantizando escalabilidad, mantenibilidad y flexibilidad para futuras expansiones.

El enfoque de esta memoria se centra en la implementación de la arquitectura multicámara, el desarrollo del frontend en React.js, la integración con el backend construido en FastAPI, y la optimización del procesamiento de video a través de técnicas de suavizado (EMA) y seguimiento (IoU tracking). Estas tareas fueron fundamentales para garantizar la estabilidad visual del sistema, su capacidad de operar en tiempo real y la correcta interacción entre la capa de presentación y la de procesamiento.

3.1. Análisis de Requerimientos

El análisis de requerimientos constituye la base sobre la cual se define la estructura funcional del sistema. Esta etapa se orienta a identificar las necesidades tanto operativas como técnicas que debía cumplir la solución, considerando las restricciones propias del entorno objetivo (Metro de Santiago) y los objetivos de escalabilidad definidos por el equipo. A partir de ello se identificaron los **Requerimientos Funcionales (RF)**, **Requerimientos No Funcionales (RNF)** y los **Requerimientos de Interfaz (RI)**.

3.1.1. Requerimientos Funcionales (RF)

- RF1: El sistema debe capturar flujos de video provenientes de cámaras IP mediante protocolo RTSP/HTTP/MJPEG.
- RF2: El sistema debe detectar personas en tiempo real utilizando un modelo de inteligencia artificial basado en YOLOv8.
- RF3: El sistema debe permitir la gestión simultánea de múltiples cámaras activas.
- RF4: El sistema debe calcular métricas de afluencia y generar alertas al superar umbrales definidos.

- RF5: El sistema debe permitir la visualización de los flujos procesados y métricas en un panel web.
- RF6: El sistema debe permitir la descarga de reportes históricos en formato CSV.
- RF7: El sistema debe autenticar a los usuarios mediante credenciales de acceso.

3.1.2. Requerimientos No Funcionales (RNF)

- RNF1: La latencia cliente-servidor del sistema debe ser menor a 500 ms por cámara.
- RNF2: El sistema debe soportar al menos cinco cámaras IP simultáneas sin pérdida significativa de rendimiento.
- RNF3: La arquitectura debe ser modular y escalable, permitiendo la incorporación de nuevos módulos y cámaras.
- RNF4: La interfaz debe ser responsiva y permitir la actualización en tiempo real de métricas.
- RNF5: El sistema debe operar en entornos locales (edge computing) sin dependencia de servicios externos.

3.1.3. Requerimientos de Interfaz (RI)

- RI1: La interfaz web debe permitir iniciar sesión mediante credenciales (usuario y contraseña).
- RI2: La interfaz debe mostrar múltiples cámaras simultáneamente, junto con sus métricas asociadas.
- RI3: La interfaz debe incluir un panel de alertas con notificaciones visuales.
- RI4: El sistema debe generar gráficos históricos de afluencia por cámara.
- RI5: El sistema debe permitir la exportación de datos en formato CSV mediante un botón de descarga.

A partir del análisis de requerimientos, se establece una trazabilidad con los módulos del sistema descritos en la sección siguiente. Los requerimientos RF3, RF5, RF6 y RI2–RI5 constituyeron el foco principal de desarrollo, pues abarcan el procesamiento multicámara, la interfaz gráfica, la visualización de alertas y la integración de reportes. Asimismo, esta memoria aborda de manera explícita todos los Requerimientos No Funcionales (RNF1–RNF5), los cuales fueron fundamentales para garantizar el rendimiento del sistema. En particular, se trabajó en la latencia cliente-servidor, el soporte estable para múltiples cámaras, la modularidad de la arquitectura, la actualización en tiempo real de métricas y la capacidad de operar de forma autónoma en entornos locales.

3.2. Diseño de la Solución

El diseño de la solución se estructura en torno a una arquitectura modular cliente-servidor, donde cada componente cumple una función específica dentro del flujo general de procesamiento. Esta división permite abordar el desarrollo de manera paralela, distribuyendo responsabilidades entre los miembros del equipo y garantizando la independencia funcional de los módulos.

3.2.1. Diagrama de Contexto

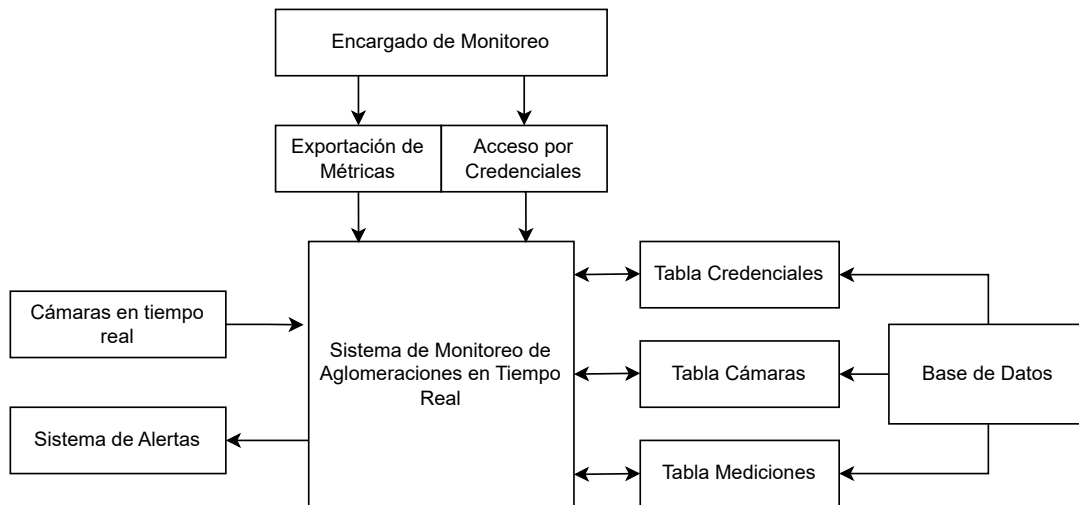


Figura 3.1: Diagrama de contexto del sistema.

El diagrama de contexto muestra la interacción entre los principales actores del sistema: las cámaras IP, el backend de procesamiento, la base de datos y la interfaz de usuario. Las cámaras capturan los flujos de video que son enviados al servidor FastAPI, donde se ejecuta el modelo YOLOv8 para detectar personas. Los resultados se almacenan temporalmente y se envían al frontend para su visualización. El usuario, mediante la interfaz web, puede monitorear el estado de las cámaras, revisar métricas de ocupación, descargar reportes o recibir alertas automáticas.

3.2.2. Diagrama de Arquitectura del Sistema

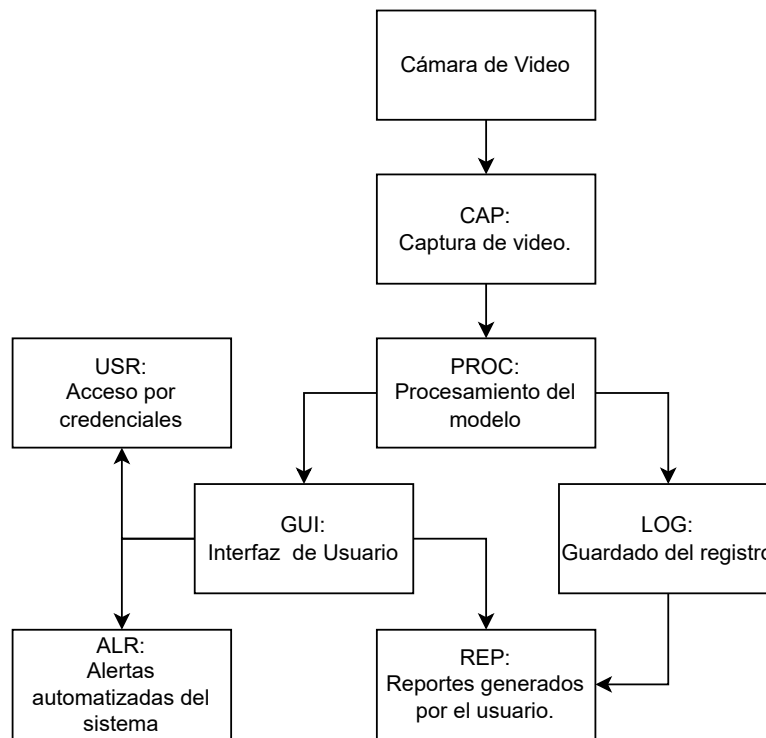


Figura 3.2: Diagrama de Arquitectura del sistema.

La arquitectura del sistema se compone de siete módulos interconectados: **CAP** (Captura), **PROC** (Procesamiento), **GUI** (Interfaz Gráfica), **LOG** (Registro), **ALR** (Alertas), **REP** (Reportes) y **USR** (Usuarios). Cada uno cumple una función bien definida dentro del flujo de trabajo y se comunica mediante servicios REST y colas internas de procesamiento.

El backend implementa un modelo concurrente, donde cada cámara se procesa en un hilo independiente a través del módulo CAP y PROC, mientras que el frontend maneja la visualización y control mediante el módulo GUI. Las alertas, reportes y registros se gestionan en módulos específicos, asegurando independencia y mantenibilidad.

3.2.3. Enumeración de Módulos

Módulo	Propósito	Sección
Módulo CAP: Captura de video (CAP)	Gestionar la conexión con cámaras IP (RTSP/MJPEG), adquirir frames y entregar flujos estandarizados al subsistema de procesamiento.	3.2.6
Módulo PROC: Procesamiento (PROC)	Ejecutar inferencia de detección (YOLOv8), ejecutar tracking por IoU, calcular métricas de ocupación y suavizar conteos mediante EMA.	3.2.7
Módulo GUI: Interfaz gráfica (GUI)	Ofrecer panel web multicámara, gráficos temporales, gestión de cámaras (alta/baja) y portal de notificaciones.	3.2.8
Módulo LOG: Registro (LOG)	Registrar eventos del sistema, errores y trazas operativas para auditoría y diagnóstico.	3.2.9
Módulo ALR: Alertas (ALR)	Evaluar umbrales configurables, agregar alertas por cámara y despachar notificaciones al frontend.	3.2.10
Módulo REP: Reportes (REP)	Generar y servir reportes históricos en CSV sobre conteos y métricas por cámara.	3.2.11
Módulo USR: Usuarios (USR)	Gestionar autenticación, roles y control de acceso mediante JWT.	3.2.12

Cuadro 3.1: Enumeración de módulos del sistema.

3.2.4. Matriz de Requerimientos y Módulos

Req. Funcional	CAP	PROC	GUI	LOG	ALR	REP	USR
RF1	X						
RF2		X					
RF3	X	X	X				
RF4		X			X		
RF5			X				
RF6						X	
RF7							X
RNF1	X	X					
RNF2	X	X	X				
RNF3	X	X	X	X	X	X	X
RNF4			X				
RNF5	X	X					

Cuadro 3.2: Matriz de trazabilidad entre requerimientos y módulos.

3.2.5. Descripción de Módulos

A continuación se detalla cada módulo del sistema, siguiendo el formato profesional utilizado en memorias anteriores. Cada tabla incluye el propósito, alcance, dependencias, supuestos, restricciones y estructura general del módulo.

3.2.6. Módulo CAP — Captura de video

Propósito	Gestionar conexiones RTSP/MJPEG a cámaras IP, proveer frames estandarizados para el procesamiento en tiempo real.
Alcance	Adquisición continua de frames de múltiples cámaras; estandarización de resolución (letterbox a 1280x720 o similar); control de alta/baja de cámaras y reconexión automática ante fallas.
Dependencias	OpenCV, ffmpeg (si aplica), threading, camera_manager.py.
Supuestos	Cámaras configuradas en la red local, disponibilidad de ancho de banda suficiente y credenciales válidas.
Restricciones	Ancho de banda, latencia de red y límites físicos del hardware para decodificación.
Estructura general	Implementado en Python con clases de gestión de cámara (<code>Camera</code> , <code>CameraManager</code>), métodos <code>start()</code> , <code>stop()</code> y reconexión.

Cuadro 3.3: Módulo CAP — Captura de video.

3.2.7. Módulo PROC — Procesamiento de imágenes

Propósito	Ejecutar inferencia de detección (YOLOv8), aplicar tracking (IoU), calcular métricas de ocupación y aplicar suavizado EMA para estabilizar conteos.
Alcance	Soporta inferencia en CPU/GPU, tracking temporal, cálculo de <code>crowd_index</code> (área segmentada / área total), ventanas estadísticas (mean, peak) y registro temporal de métricas.
Dependencias	Ultralytics (YOLOv8), NumPy, OpenCV, PyTorch (si aplica), cudatoolkit (opcional).
Supuestos	Modelos pre-entrenados logran detección de clase <i>person</i> adecuada en condiciones operativas.
Restricciones	Memoria GPU/CPU limitada, variaciones de iluminación y oclusiones parciales.
Estructura general	Clase <code>VideoProcessor</code> por cámara que corre en hilo independiente, con buffer de frames, cola de resultados y funciones de post-procesado.

Cuadro 3.4: Módulo PROC — Procesamiento de imágenes.

3.2.8. Módulo GUI — Interfaz Gráfica

Propósito	Ofrecer panel web con vista multicámara, gráficos temporales, gestión de cámaras, alertas y opciones de exportación.
Alcance	Componentes React para login, grid de cámaras, administración de cámaras, panel de alertas y descarga de reportes.
Dependencias	React, Axios, Recharts/Chart.js, portal de notificaciones, CSS y librerías de UI (opcional).
Supuestos	Backend expone endpoints REST y streaming MJPEG estables; cliente dispone de navegador moderno.
Restricciones	Rendimiento del navegador y latencia de la red.
Estructura general	Componentes: <code>Login.jsx</code> , <code>CamaraManager.jsx</code> , <code>CamaraGrid.jsx</code> , <code>Dashboard.jsx</code> , <code>Alerts.jsx</code> ; almacenamiento local del JWT y hooks para refresco.

Cuadro 3.5: Módulo GUI — Interfaz Gráfica.

3.2.9. Módulo LOG — Registro de eventos

Propósito	Registrar eventos operativos, errores y metadatos para auditoría, diagnóstico y trazabilidad.
Alcance	Logging local y opcional persistencia en BD; niveles (INFO, WARNING, ERROR); rotación y retención configurables.
Dependencias	Python logging, SQLAlchemy (si se persiste), herramientas de monitorización (opcional).
Supuestos	Espacio en disco suficiente y políticas de retención definidas.
Restricciones	Uso de I/O y espacio; potencial impacto en rendimiento si el logging es excesivo.
Estructura general	Módulo central de logging integrado con FastAPI y con puntos de instrumentación en CAP/PROC/ALR.

Cuadro 3.6: Módulo LOG — Registro de eventos.

3.2.10. Módulo ALR — Alertas

Propósito	Detectar condiciones de sobreocupación o anomalías en métricas y despachar alertas al front-end con metadatos (cam_id, timestamp, count, occupancy).
Alcance	Evaluación de umbrales configurables, cooldown anti-spam, agregación y normalización de alertas.
Dependencias	Configuración (.env), camera_manager, endpoints /alerts y frontend Alerts.jsx.
Supuestos	Umbrales y parámetros configurados por operadores; métricas entregadas por PROC de forma fiable.
Restricciones	Falsos positivos por errores de detección; riesgo de duplicación sin cooldown adecuado.
Estructura general	Servicio backend que filtra, agrega y expone alertas; frontend con deduplicación y backoff de polling.

Cuadro 3.7: Módulo ALR — Alertas.

3.2.11. Módulo REP — Reportes

Propósito	Generar reportes históricos en CSV que permitan análisis posterior de flujo y ocupación.
Alcance	Exportación por rango temporal, por cámara o global; endpoints protegidos para descarga.
Dependencias	Pandas, FastAPI static files, persistencia temporal o en BD.
Supuestos	Datos métricos disponibles y consistentes en memoria o BD.
Restricciones	Volumen de datos y ventana máxima de extracción por petición.
Estructura general	Funciones backend que construyen DataFrame y devuelven CSV a la GUI.

Cuadro 3.8: Módulo REP — Reportes.

3.2.12. Módulo USR — Usuarios

Propósito	Administrar autenticación y autorización de usuarios para proteger el acceso a la plataforma.
Alcance	Login, emisión y verificación de JWT, roles básicos (operador, administrador) y endpoints /auth/*.
Dependencias	FastAPI, jose JWT, SQLAlchemy y base de datos de usuarios.
Supuestos	Existencia de usuarios registrados y credenciales válidas.
Restricciones	Tiempo de expiración de tokens y políticas de seguridad.
Estructura general	Endpoints de autenticación, middleware para verificación y almacenamiento seguro de credenciales.

Cuadro 3.9: Módulo USR — Usuarios.

3.2.13. Foco de Módulos en la Memoria

El desarrollo de esta memoria se centra en los módulos CAP, PROC, GUI y ALR, los cuales conforman el núcleo operativo del sistema y dan soporte a las funcionalidades principales de captura, procesamiento, visualización y notificación. El módulo **CAP** habilita la adquisición y gestión de múltiples flujos RTSP/HTTP/MJPEG mediante hilos dedicados, asegurando una captura estable y con reconexión automática. El módulo **PROC** implementa el procesamiento concurrente de video, integrando detección con YOLOv8, cálculo de métricas, tracking por IoU y suavizado temporal. En el módulo **GUI**, se diseña e implementa la interfaz completa en React.js, la cual permite visualizar métricas en tiempo real, administrar cámaras y gestionar reportes. Finalmente, el módulo **ALR** integra la lógica de alertas, el polling adaptativo, la deduplicación y la presentación visual mediante toasts. Estos cuatro módulos constituyen la base técnica del sistema y representan el alcance directo del trabajo desarrollado en esta memoria.

3.3. Implementación

La implementación del sistema corresponde a la fase en la cual las decisiones de diseño, la arquitectura definida y los módulos conceptualizados se materializan en componentes concretos, funcionales y articulados entre sí. Esta sección describe de manera detallada el desarrollo del backend, el frontend, la lógica multicámara, la integración entre capas, el sistema de métricas y alertas, las optimizaciones aplicadas y el proceso de pruebas y validación. La orientación de esta implementación refleja el foco de esta memoria: la construcción del frontend, la integración completa con el backend, la lógica multicámara, la gestión de alertas y la optimización del procesamiento y visualización.

3.3.1. Objetivo Técnico de la Implementación

El objetivo central de la implementación consiste en construir un sistema capaz de monitorear en tiempo real la afluencia de personas mediante múltiples cámaras simultáneas, procesando cada flujo de video por separado y exponiendo sus resultados a través de una interfaz web dinámica. Para ello, se requiere una infraestructura capaz de manejar concurrencia intensiva, realizar inferencia sobre imágenes de manera continua, comunicar resultados mediante una API eficiente y presentar dichos resultados al usuario final con claridad y baja latencia.

De forma estricta, el sistema debe cumplir tres ejes funcionales: (1) capturar los flujos de videovigilancia y procesarlos mediante un modelo inteligente (usado como caja negra) para obtener el conteo de personas y su nivel de ocupación; (2) coordinar múltiples instancias independientes de procesamiento —una por cámara— en paralelo y sin bloquear el sistema; y (3) presentar al usuario una interfaz intuitiva, responsiva y actualizada en tiempo real que permita gestionar cámaras, visualizar conteos, revisar alertas y descargar reportes históricos. La consecución de estos objetivos requiere una integración coherente entre backend y frontend, así como una capa intermedia de optimización que habilite estabilidad en el procesamiento y en la experiencia de usuario.

3.3.2. Arquitectura General Implementada

La arquitectura final se implementó siguiendo un enfoque cliente-servidor, donde el backend desarrollado en **FastAPI** asume la responsabilidad de capturar, procesar y exponer los datos, mientras que el frontend construido en **React.js** se encarga de la visualización, interacción y administración del sistema. Cada cámara se procesa en un módulo completamente aislado mediante un hilo dedicado, lo cual habilita escalabilidad horizontal sin que el procesamiento de una cámara afecte a las demás.

Este diseño permite combinar procesamiento intensivo con una interfaz fluida, dado que la carga de trabajo se encuentra distribuida. La comunicación se realiza mediante endpoints REST para datos estructurados (JSON) y mediante flujos MJPEG, RTSP, HTTP y rutas locales para los streams de video. Adicionalmente, se utilizan estructuras en memoria en el backend para almacenar estados, métricas y alertas sin generar sobrecarga en la base de datos.

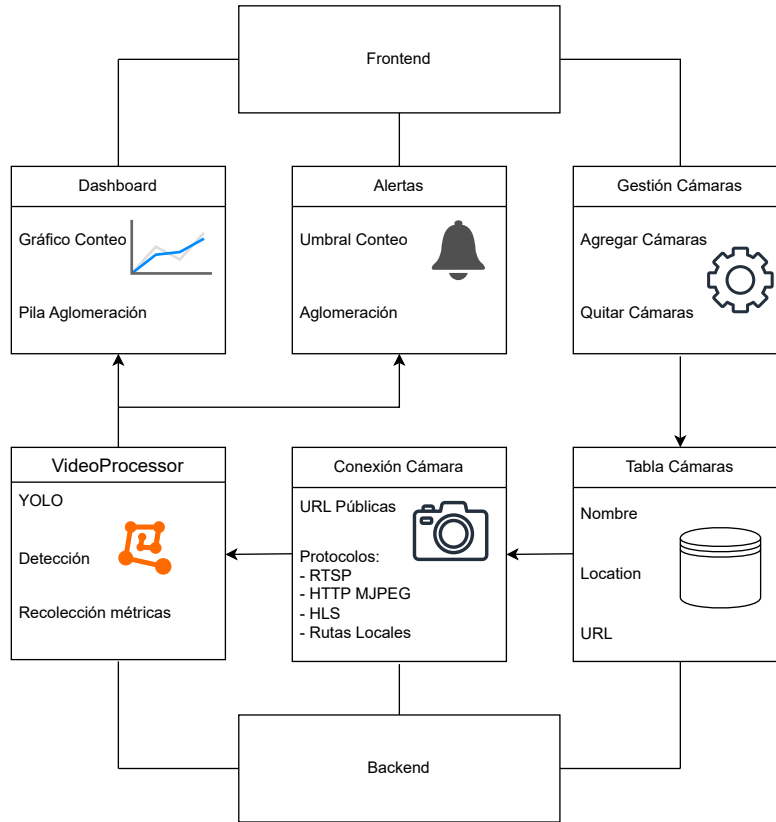


Figura 3.3: Estructuración de componentes del sistema.

3.3.3. Implementación del Backend

El backend constituye la columna vertebral del sistema y materializa los módulos CAP, PROC, LOG y ALR del diseño original. La implementación fue construida principalmente sobre tres componentes fundamentales: `main.py`, `camera_manager.py` y `video_processor.py`. En conjunto, estos elementos permiten administrar cámaras, ejecutar el procesamiento en paralelo, exponer métricas al frontend y gestionar las alertas desencadenadas por umbrales configurados.

Rutas y Servicios Principales

En `main.py` se definieron los endpoints encargados de recibir solicitudes del frontend y entregar datos en tiempo real. Esta capa funciona como una abstracción sobre el procesamiento interno, presentando una interfaz limpia para las operaciones del sistema. Entre las rutas más relevantes se encuentran:

- `/cameras`: administra adición, edición y eliminación de cámaras.
- `/stream/{cam_id}`: expone el flujo procesado en formato MJPEG, RTSP, HTTP y rutas locales.
- `/metrics`: entrega las métricas de todas las cámaras activas.
- `/metrics/{cam_id}`: devuelve métricas específicas de una cámara.

- `/alerts`: entrega las alertas que se han generado.
- `/report/{cam_id}`: genera reportes CSV con el historial de la cámara.

Cada endpoint fue implementado con la premisa de minimizar latencia y sobrecarga. Por ejemplo, las métricas se actualizan en memoria dentro de cada `VideoProcessor`, lo que permite consultas rápidas sin consultas a la base de datos.

Orquestación Multicámara (`camera_manager.py`)

La orquestación multicámara constituye el núcleo de la implementación. El componente `camera_manager.py` mantiene un diccionario global que asocia cada identificador de cámara con su correspondiente instancia de `VideoProcessor`. Esto permite gestionar las cámaras de forma dinámica: agregar, eliminar o reconfigurar cámaras sin detener el sistema.

Su responsabilidad incluye:

- Creación y destrucción de procesadores por cámara.
- Aplicación de umbrales personalizados por cámara.
- Protección de concurrencia mediante `threading.Lock`.
- Exposición de procesadores al resto del backend.

El siguiente pseudocódigo resume la función principal de obtención de procesadores:

```
function get_processor(cam_id):
    if cam_id not in processors:
        lock.acquire()
        if cam_id not in processors:
            processor = VideoProcessor(cam_id)
            processors[cam_id] = processor
            processor.start()
        lock.release()
    return processors[cam_id]
```

Este comportamiento de “instanciación perezosa” reduce el uso de recursos y evita inicializar cámaras no utilizadas.

Procesamiento por Cámara (`video_processor.py`)

Cada instancia de `VideoProcessor` ejecuta un pipeline completo de procesamiento, el cual incluye captura, inferencia, cálculo de métricas y emisión del flujo procesado. En términos de ingeniería, este componente constituye la unidad fundamental de trabajo concurrente del sistema.

El pipeline implementado por cada procesador sigue los siguientes pasos:

1. Apertura del flujo mediante RTSP/FFmpeg.
2. Captura continua de frames utilizando OpenCV.
3. Aplicación del modelo de detección YOLOv8 (caja negra).

4. Conteo de personas.
5. Generación del índice de ocupación (*crowd index*).
6. Aplicación de suavizado EMA a las métricas.
7. Generación del frame anotado en JPEG para el stream MJPEG, RTSP, HTTP y rutas locales.
8. Incorporación de alertas según los umbrales definidos.

El siguiente pseudocódigo captura el bucle principal:

```
while running:
    frame = capture.read()

    if should_infer():
        detections = model(frame)
        tracked = tracker.update(detections)
        count = len(tracked)
        crowd = compute_crowd_index(detections)

    metrics.update(EMA(count, crowd, timings))
    alerts.check(count, crowd)

    annotated = draw_boxes(frame, tracked)
    output_queue.push(encode_jpeg(annotated))
```

Este bucle ejecuta decenas de veces por segundo y representa el componente computacionalmente más intensivo del sistema.

3.3.4. Implementación del Frontend

El frontend fue desarrollado íntegramente para esta memoria, constituyendo el aporte más significativo desde el punto de vista de interfaz gráfica, usabilidad e integración. El sistema fue construido en React.js empleando componentes funcionales, hooks de estado y Axios para la comunicación con el backend. Cada vista fue diseñada para mantener independencia lógica y permitir una navegación fluida.

Gestión de Cámaras

El componente `CamaraManager.jsx` implementa un formulario completo que permite registrar cámaras nuevas, editar configuraciones existentes y establecer umbrales personalizados de alerta. La lógica interna integra validación de campos, feedback visual y envío del payload al endpoint `/cameras`.

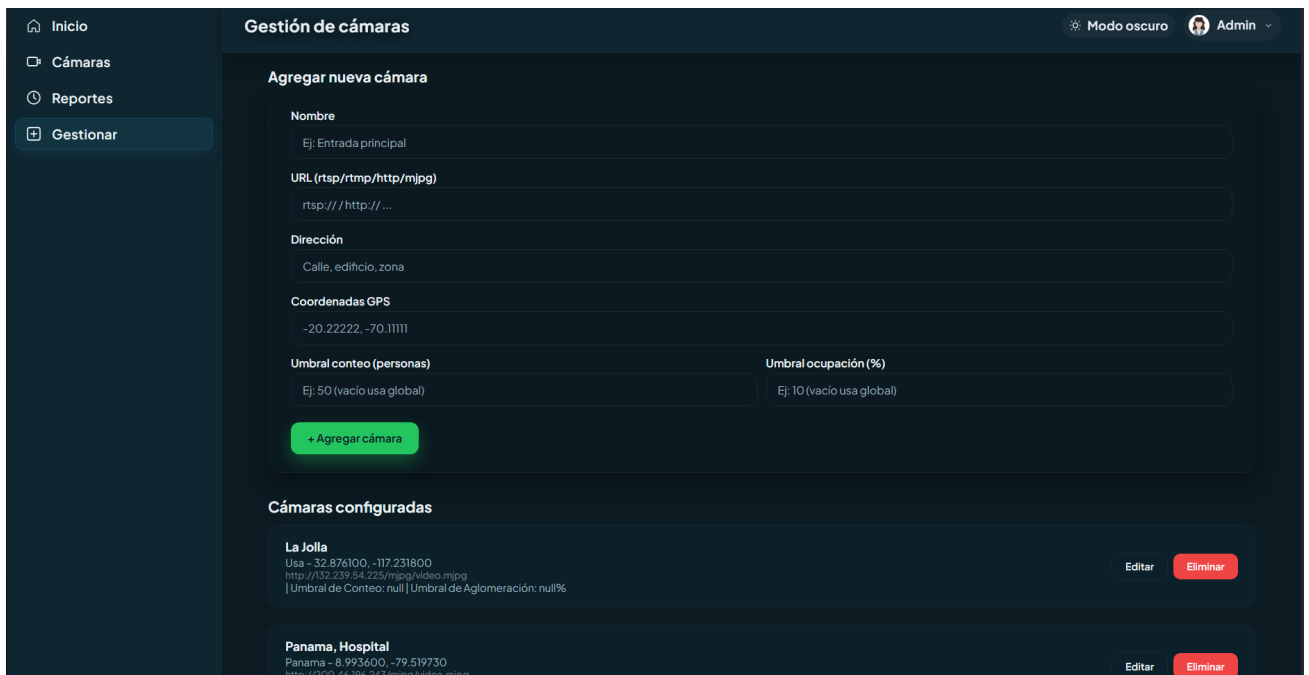


Figura 3.4: Vista de Gestión de Cámaras.

El formulario permite configurar:

- Identificador público de la cámara.
- Nombre y ubicación.
- URL RTSP, RTMP, HTTP y rutas locales.
- Coordenadas para métricas futuras.
- Umbral máximo de conteo.
- Umbral de ocupación (%).

Dashboard Multicámara

La vista principal del sistema corresponde al dashboard implementado en `CamaraGrid.jsx`. Esta sección muestra todas las cámaras gestionadas, renderizando para cada una un elemento independiente que incluye:

- El flujo de video procesado mediante el endpoint `/stream/{cam_id}?fps=8`.
- Las métricas en tiempo real (FPS, latencias de inferencia y captura, conteo, ocupación).
- Indicadores visuales de estado.

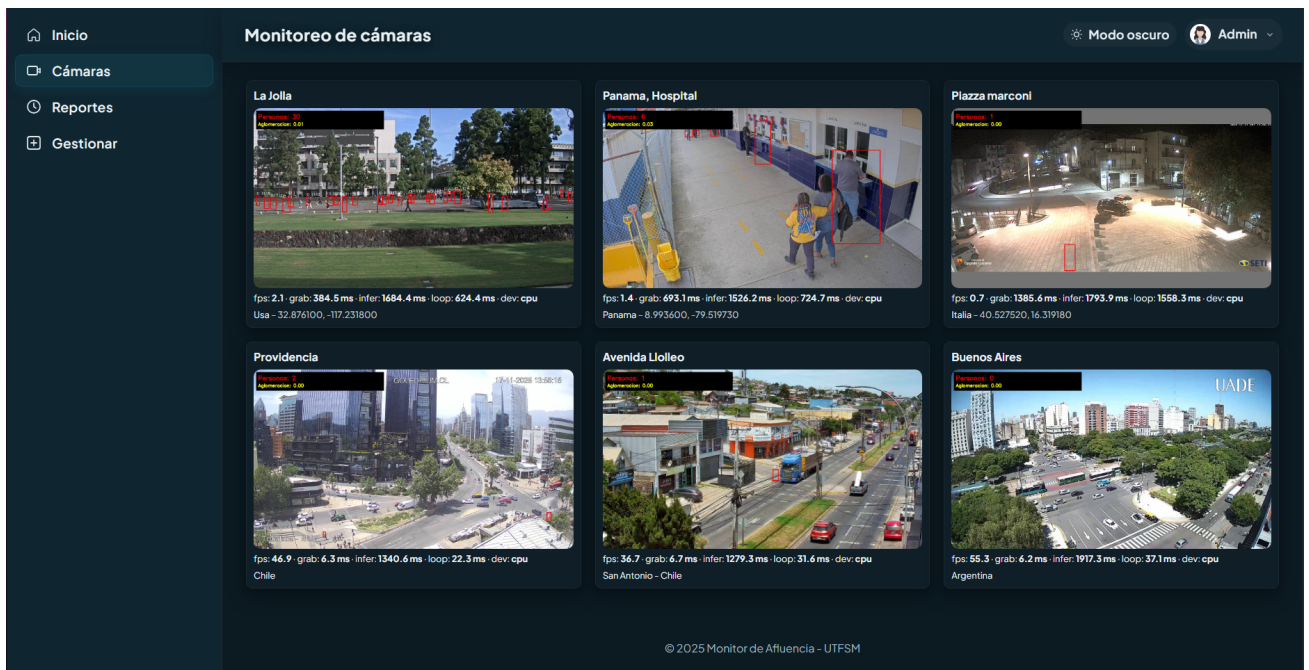


Figura 3.5: Vista de Multicámara.

Para mantener las métricas actualizadas sin interrumpir la fluidez de la interfaz, se implementó un mecanismo de actualización denominado *polling periódico*. Este mecanismo consiste en que el frontend solicita información al backend cada cierto intervalo de tiempo, en lugar de mantener una conexión permanente. En este caso, el componente React envía una petición cada dos segundos al endpoint `/metrics/{cam_id}` para obtener los valores más recientes de conteo, latencias, FPS y nivel de ocupación.

El *polling* resulta especialmente útil cuando la arquitectura no utiliza canales de comunicación persistentes como WebSockets. En su lugar, el cliente realiza solicitudes periódicas y controladas, preguntando explícitamente si existe información nueva. Esto permite mantener la interfaz liviana, reducir la complejidad de la comunicación y evitar saturar al servidor con solicitudes excesivas.

El siguiente pseudocódigo ilustra el funcionamiento del mecanismo:

```
useEffect(() => {
  const interval = setInterval(() => {
    axios.get(`/metrics/${cam_id}`).then(res => {
      setMetrics(res.data);
    });
  }, 2000);

  return () => clearInterval(interval);
}, [cam_id]);
```

Gracias a este proceso periódico, el dashboard logra mantenerse sincronizado con las métricas generadas en tiempo real por cada cámara, preservando fluidez y estabilidad visual sin producir sobrecarga en la red o en el servidor.

Sistema de Alertas

El componente `Alerts.jsx` implementa un sistema de notificaciones no intrusivas basado en toasts, los cuales se despliegan en la esquina inferior derecha del navegador y permiten informar al operador cuando una cámara supera los umbrales configurados de ocupación o conteo. Cada alerta se genera con su identificador, su cámara asociada y el timestamp correspondiente, garantizando que el usuario disponga de información clara y contextualizada sobre el evento detectado.

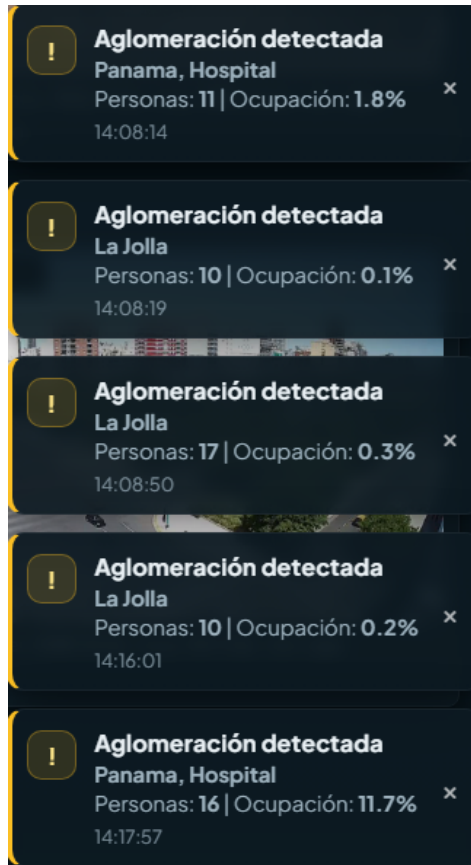


Figura 3.6: Vista de alertas.

El sistema opera mediante un mecanismo de consulta periódica (*polling*) al endpoint `/alerts`, combinado con una política de *backoff progresivo*. Este enfoque permite ajustar dinámicamente la frecuencia de sondeo según la actividad reciente: cuando se detectan alertas nuevas, el intervalo de consulta se mantiene reducido (por ejemplo, 2 segundos); en ausencia de actividad prolongada, el intervalo aumenta gradualmente hasta un valor máximo, reduciendo la carga sobre el backend y evitando sondeos innecesarios. Esta estrategia equilibra adecuadamente oportunidad, eficiencia y estabilidad en sesiones prolongadas de monitoreo.

Para prevenir la repetición de eventos, el cliente implementa un mecanismo de deduplicación basado en identificadores únicos y verificación temporal, asegurando que un mismo evento no sea presentado más de una vez incluso si el backend lo reporta en ciclos consecutivos del polling. El endpoint `/alerts` puede además filtrar temporalidades recientes, pero la responsabilidad principal de evitar duplicaciones recae en la lógica del cliente.

Como complemento a la deduplicación, el sistema incorpora una política explícita de tiempo de

vida (*Time-To-Live*, TTL) para cada notificación. Cada toast permanece visible por un máximo de **10 minutos** desde el momento de su creación y se cierra automáticamente una vez transcurrido dicho periodo, aun cuando el operador no lo haya descartado manualmente. Este mecanismo evita la acumulación de alertas antiguas, mantiene la interfaz visualmente despejada y asegura que el operador centre su atención en eventos recientes y operativamente relevantes.

Finalmente, la implementación incluye un comportamiento de degradación para condiciones inestables: si el servidor presenta errores consecutivos o se detectan incrementos anómalos de latencia, el cliente amplía los intervalos de sondeo y registra métricas locales de fallos e intentos, preservando la continuidad del sistema de alertas sin sobrecargar la infraestructura. En conjunto, el polling adaptativo, el backoff progresivo, la deduplicación por identificadores y el TTL de 10 minutos conforman un sistema de notificaciones robusto, eficiente y coherente con los requisitos operativos del monitoreo multicámara.

El siguiente pseudocódigo ilustra el funcionamiento del mecanismo:

```
# Mecanismo de polling adaptativo con backoff, deduplicación y TTL
initialize:
    interval = 2                # intervalo mínimo (segundos)
    max_interval = 30           # intervalo máximo para backoff
    last_alert_ids = set()      # IDs ya mostrados
    TTL = 600                   # 10 minutos en segundos

loop forever:
    current_time = now()

    # Solicitar alertas recientes al backend
    response = GET("/alerts")
    alerts = response.data

    new_alerts = []
    for alert in alerts:
        id = alert.id
        ts = alert.timestamp

        # Descartar si ya fue presentada (deduplicación)
        if id in last_alert_ids:
            continue

        # Validar temporalidad (alertas muy antiguas no se muestran)
        if current_time - ts > TTL:
            continue

        new_alerts.append(alert)
        last_alert_ids.add(id)
        show_toast(alert)      # Render en interfaz
```

```

# Autocierre de toasts por TTL
for toast in active_toasts:
    if current_time - toast.timestamp >= TTL:
        close_toast(toast)

# Ajustar intervalo de polling según actividad
if len(new_alerts) > 0:
    interval = 2    # actividad reciente → polling rápido
else:
    interval = min(interval * 2, max_interval)    # backoff progresivo

sleep(interval)

```

3.3.5. Integración FastAPI–React

La integración entre el backend y el frontend se basa principalmente en Axios para el envío y recepción de datos en formato JSON. A nivel de streams, la integración se realiza directamente desde la etiqueta `<img>` de HTML, consumiendo el endpoint MJPEG sin necesidad de bibliotecas externas.

El siguiente pseudocódigo ejemplifica el ciclo de actualización del frontend:

```

useEffect(() => {
    async function fetchMetrics() {
        const res = await axios.get(`/metrics/${cam_id}`);
        setMetrics(res.data);
    }
    const interval = setInterval(fetchMetrics, 2000);
    return () => clearInterval(interval);
}, [cam_id]);

```

3.3.6. Sistema de Métricas y Alertas

El sistema de métricas integra la información generada por cada instancia de `VideoProcessor`, agregándola de forma estructurada y exponiéndola a través de endpoints dedicados. Este mecanismo permite que el frontend acceda periódicamente a indicadores clave de operación, tales como el número de personas detectadas, la ocupación estimada, las latencias del sistema y los FPS efectivos de cada cámara. Las métricas consideradas en esta etapa incluyen:

- FPS efectivos.
- Latencias de captura e inferencia.
- Conteo de personas.
- Índice de ocupación.

Índice de Ocupación.

Dentro de estas métricas, el índice de ocupación (o *crowd index*) constituye un indicador fundamental para estimar cuánta proporción del campo visual está ocupada por personas. A diferencia del conteo absoluto, esta métrica captura la densidad visual del entorno, permitiendo identificar escenarios donde la congestión es elevada aunque el número de individuos no varíe significativamente.

En la implementación del sistema, el cálculo del índice de ocupación se basa prioritariamente en las máscaras de segmentación generadas por YOLOv8 en su variante *seg*. Dichas máscaras permiten identificar de forma precisa los píxeles pertenecientes a la clase “person”, proporcionando una estimación detallada del área efectivamente ocupada. El procedimiento seguido para cada frame es el siguiente:

1. Se seleccionan las detecciones correspondientes a la clase “person”.
2. Si el modelo entrega máscaras válidas, estas se reescalan al tamaño del frame y se calcula el número de píxeles activos por máscara.
3. Se suman las áreas activas de todas las máscaras de persona y se normaliza dividiendo por el área total del frame (ancho $\times$ alto).

La formulación matemática es:

$$\text{Índice de ocupación} = \frac{\sum_{i=1}^N \text{área}(\text{máscara}_i)}{\text{área total del frame}}$$

donde cada $\text{área}(\text{máscara}_i)$ corresponde al conteo de píxeles asignados a la i -ésima máscara de persona.

Nota técnica y alcance del método. Si bien este procedimiento permite obtener un valor operativo y estable, su precisión es limitada para un análisis riguroso de densidad poblacional. La métrica depende fuertemente de la calidad de la segmentación, del ángulo de la cámara y de la escala aparente de los individuos en la imagen, lo que restringe su utilidad en escenarios complejos. Es importante señalar que este indicador constituye un **prototipo inicial** incorporado únicamente para efectos demostrativos. El enfoque de esta memoria no se orienta al desarrollo de métricas avanzadas de ocupación, sino a la integración y visualización del sistema; por ello, se limita a mostrar el valor generado por el backend sin profundizar en optimizaciones metodológicas. La discusión sobre alternativas más robustas se retoma en el capítulo de Conclusiones y Trabajo Futuro.

Fallback con bounding boxes.

En ausencia de máscaras —ya sea por una variante del modelo sin segmentación o fallos puntuales en la inferencia— el sistema utiliza un mecanismo alternativo basado en bounding boxes. En ese caso, el área ocupada se aproxima mediante:

$$\text{Índice de ocupación (aprox.)} = \frac{\sum_{i=1}^N \text{área}(\text{bbox}_i)}{\text{área total del frame}}$$

Este método garantiza continuidad operativa, pero su precisión es sustancialmente inferior debido a que la caja rectangular sobreestima el área real ocupada y es sensible a variaciones de escala y perspectiva.

Limitaciones y carácter de prototipo. El uso de *bounding boxes* como aproximación no se considera una estrategia óptima para estimar ocupación, ya que genera sesgos sistemáticos y mediciones poco representativas en escenas reales. Su incorporación responde únicamente a la necesidad de mantener un flujo funcional en el prototipo cuando la segmentación no está disponible. Tal como se especifica en el alcance del proyecto, el objetivo de esta memoria no es diseñar un estimador avanzado de ocupación, sino integrar y presentar la información generada por el backend.

Suavizado temporal.

Para evitar fluctuaciones entre frames consecutivos, el índice de ocupación se filtra mediante un Exponential Moving Average (EMA):

$$EMA_t = \alpha \cdot x_t + (1 - \alpha) \cdot EMA_{t-1}$$

Este suavizado mejora la estabilidad del indicador y permite una interpretación más robusta en contextos de alta variabilidad visual.

Alertas basadas en métricas.

El sistema de alertas utiliza tanto el conteo absoluto como el índice de ocupación suavizado para evaluar condiciones críticas. Cuando alguna de estas métricas supera los umbrales definidos por el usuario, se genera una alerta que es almacenada en una cola interna y posteriormente expuesta mediante el endpoint `/alerts`. Este mecanismo permite notificar al frontend ante situaciones de sobreocupación o anomalías en tiempo real.

3.3.7. Optimización del Rendimiento

Durante la implementación se identificaron diversos cuellos de botella relacionados con la captura RTSP, la inferencia y la visualización. Para enfrentar estos problemas se aplicaron las siguientes optimizaciones:

- **Throttle de FPS:** reducción del flujo solicitado desde el frontend utilizando parámetros `?fps=8`.
- **Downscaling:** reducción de la resolución de entrada antes de la inferencia.
- **EMA:** suavizado exponencial para estabilizar métricas ruidosas.
- **Caching interno:** reutilización de cajas de detección entre inferencias.
- **Control de hilos:** cada procesador se ejecuta como daemon, permitiendo cierre limpio.

3.3.8. Pruebas Unitarias

Las pruebas unitarias constituyen un componente esencial en la validación de sistemas modulares, ya que permiten verificar el funcionamiento correcto e independiente de cada una de las partes que componen la arquitectura general antes de proceder a su integración. En el contexto de esta memoria, se evaluaron los módulos CAP, PROC, GUI y ALR, examinando su comportamiento bajo condiciones controladas, generando evidencia concreta de su funcionamiento y utilizando criterios objetivos de aceptación. Estas pruebas garantizan que cada módulo cumpla su propósito técnico dentro del flujo multicámara, evitando que errores locales afecten el desempeño global del sistema.

Pruebas al Módulo CAP (Captura de Video)

El módulo **CAP** es responsable de la adquisición continua de frames desde múltiples fuentes de video (RTSP, HTTP y MJPEG), constituyendo el punto de entrada del pipeline de procesamiento del sistema. Dada su posición crítica dentro de la arquitectura, este módulo debe operar de manera robusta frente a variaciones en la latencia de red, fluctuaciones de bitrate y eventuales retardos en la entrega de frames, sin comprometer la continuidad del flujo ni la estabilidad temporal del procesamiento.

Con el objetivo de validar su correcto funcionamiento, se diseñaron pruebas unitarias orientadas a verificar tanto la obtención sostenida de frames válidos como la estabilidad temporal de la captura. En una primera instancia, se evaluó la capacidad del módulo para leer una secuencia continua de frames desde una fuente MJPEG, ejecutando cinco lecturas consecutivas mediante el método `read()` y comprobando que ninguno de los frames retornados fuera nulo:

```
frames = [camera.read() for _ in range(5)]
assert all(f is not None for f in frames)
```

La ejecución de esta prueba confirma que el módulo CAP mantiene una adquisición continua y confiable de imágenes, condición indispensable para el procesamiento concurrente del sistema multicámara.

Adicionalmente, y con el fin de atender la retroalimentación relativa a la estabilidad de latencia del módulo, se incorporó una validación explícita del tiempo de llegada entre frames consecutivos. Para ello, se instrumenta el módulo de captura de manera que sobreimprima directamente en cada frame información relevante de diagnóstico, incluyendo: la latencia inter-frame medida localmente, el identificador secuencial del frame y el tiempo transcurrido desde el inicio de la prueba.

La latencia de captura se calcula como la diferencia temporal entre la llegada de dos frames consecutivos, definida como $\Delta t = t_i - t_{i-1}$ y expresada en milisegundos. Dado que el primer frame no posee un precedente temporal, su latencia se fija explícitamente en 0,0 ms y se rotula como *inicio*, evitando así introducir valores atípicos asociados al arranque de la conexión o al establecimiento inicial del flujo.

La Figura 3.7 presenta la secuencia completa de cinco frames capturados desde la cámara La Jolla, donde se observa que, tras el frame inicial, las latencias medidas entre los frames 2 a 5 se mantienen dentro de un rango acotado, aproximadamente entre 380 ms y 430 ms. Esta consistencia visual y numérica evidencia que el módulo CAP entrega un flujo temporalmente estable bajo condiciones normales de adquisición MJPEG.

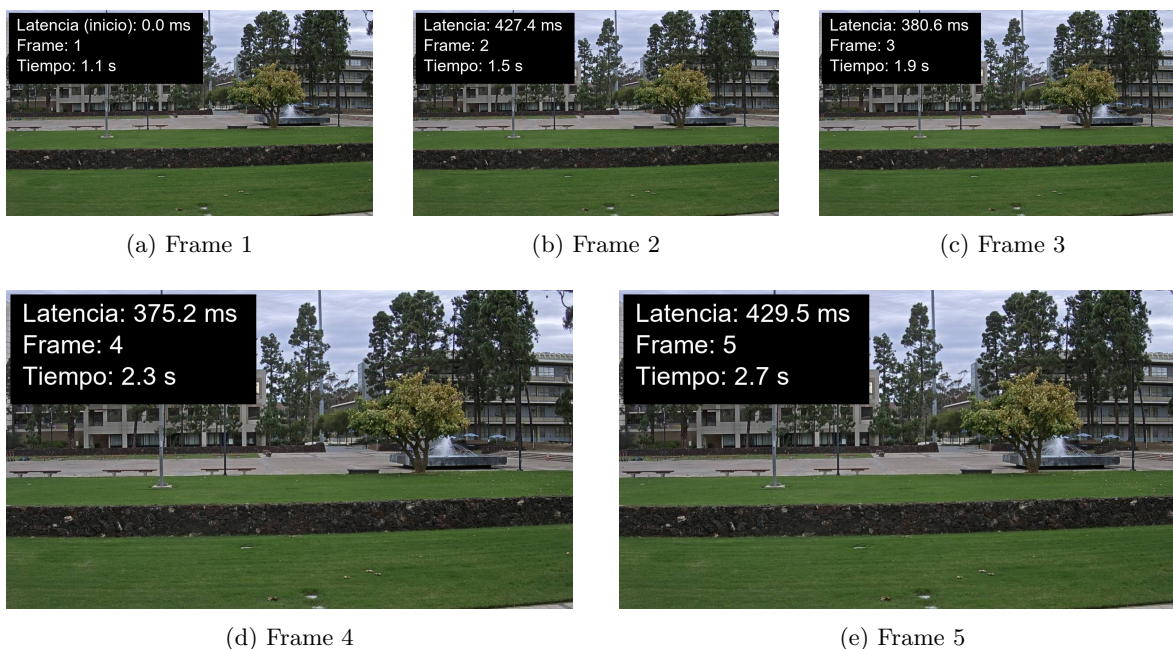


Figura 3.7: Secuencia de cinco frames consecutivos válidos desde fuente MJPEG (La Jolla), con sobreimpresión de latencia inter-frame, identificador de frame y tiempo transcurrido.

En conjunto, los resultados de esta prueba permiten concluir que el módulo CAP no solo garantiza la entrega continua de frames válidos, sino que además presenta una latencia de captura estable entre frames consecutivos, una propiedad fundamental para asegurar el correcto funcionamiento del pipeline de procesamiento y la estabilidad global del sistema multicámara.

Pruebas al Módulo PROC (Procesamiento y Métricas)

El módulo **PROC** implementa el núcleo del procesamiento del sistema, siendo responsable de la ejecución del modelo YOLOv8, la obtención de detecciones, el seguimiento de objetos mediante IoU y la generación de métricas derivadas, incluyendo el índice de ocupación y su suavizado temporal mediante un promedio móvil exponencial (EMA). Dado que este módulo concentra operaciones matemáticas y lógicas críticas, las pruebas unitarias se orientan a verificar la coherencia interna de los cálculos y la estabilidad del pipeline bajo condiciones controladas.

Las pruebas unitarias del módulo PROC se realizan utilizando **datos sintéticos** (máscaras binarias, señales simuladas y bounding boxes artificiales), con el objetivo de aislar el comportamiento de cada componente del procesamiento y evitar interferencias propias de la adquisición de video real, tales como latencia de red, variabilidad de iluminación o fluctuaciones del modelo de detección. La validación del funcionamiento integrado del sistema con flujos reales se presenta posteriormente en el capítulo de Validación mediante el Panel del sistema.

En primer lugar, se evalúa el cálculo del índice de ocupación a partir de máscaras sintéticas con áreas conocidas. Esta prueba no busca validar la métrica como un estimador preciso de densidad poblacional, sino comprobar que el procedimiento implementado es matemáticamente consistente y produce valores reproducibles a partir de una entrada controlada:

```

mask = synthetic_mask()                                # matriz binaria conocida
expected_area = mask.sum() / mask.size
computed = compute_crowd_index(mask)
assert abs(computed - expected_area) < 0.01

```

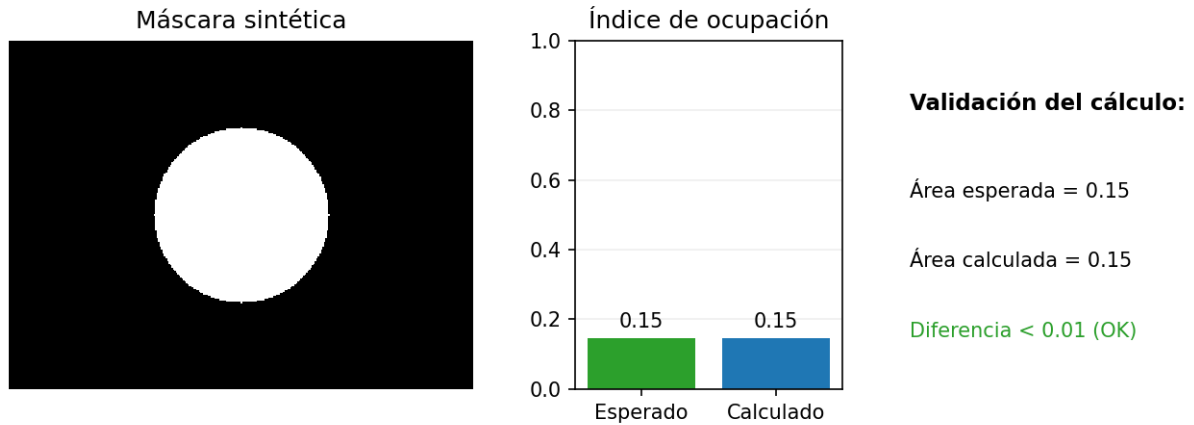


Figura 3.8: Comparación entre una máscara sintética y el índice de ocupación calculado por el módulo PROC.

Este resultado confirma que el cálculo implementado responde correctamente a variaciones controladas en el área ocupada, sin introducir inconsistencias numéricas. Como se mencionó anteriormente, el índice de ocupación constituye un indicador prototípico incorporado con fines demostrativos y no una métrica avanzada de análisis poblacional.

A continuación, se valida el mecanismo de seguimiento de objetos basado en IoU. Para ello, se emplean bounding boxes sintéticas con desplazamientos reducidos entre frames consecutivos, verificando que detecciones con alta superposición mantengan un identificador consistente:

```

box_t0 = [100, 100, 150, 200]
box_t1 = [105, 102, 150, 200]
assert track_id(box_t0) == track_id(box_t1)

```

Esta prueba permite verificar el correcto funcionamiento del algoritmo de asociación temporal sin depender de detecciones reales, asegurando estabilidad del tracking ante desplazamientos controlados.

Finalmente, se evalúa el comportamiento del suavizado temporal aplicado sobre señales de ocupación con alta variabilidad. Para ello, se genera una señal sintética ruidosa que simula fluctuaciones abruptas y se compara con la señal resultante tras aplicar el filtro EMA:

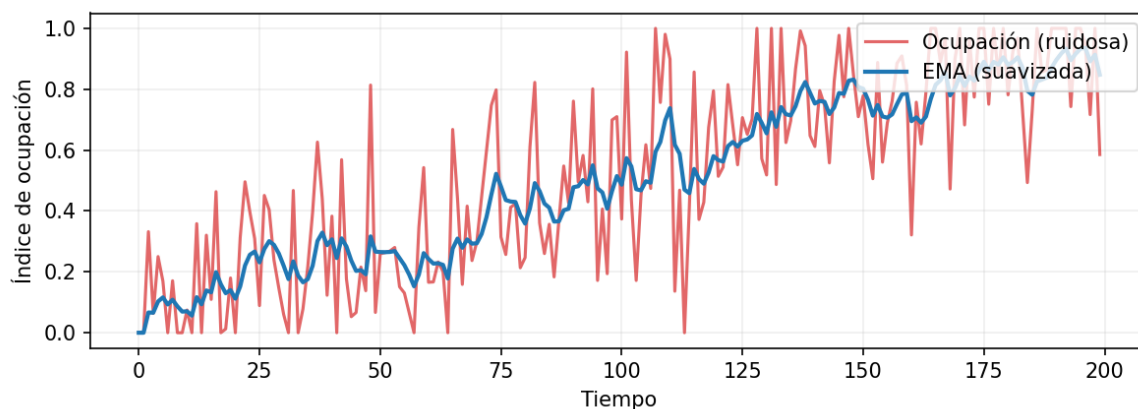


Figura 3.9: Comparación entre señal de ocupación sintética original y señal suavizada mediante EMA.

La evidencia gráfica muestra una reducción clara de variaciones abruptas, confirmando que el mecanismo de suavizado atenúa el ruido temporal y mejora la estabilidad visual de la métrica presentada al usuario.

En conjunto, las pruebas unitarias realizadas demuestran que el módulo PROC implementa correctamente los mecanismos fundamentales de procesamiento, seguimiento y generación de métricas, entregando resultados coherentes y estables bajo condiciones controladas. La evaluación del comportamiento del módulo frente a flujos reales de video se aborda en el capítulo de Validación, donde se analiza el desempeño del sistema completo en escenarios multicámara.

Pruebas al Módulo GUI (Interfaz React.js)

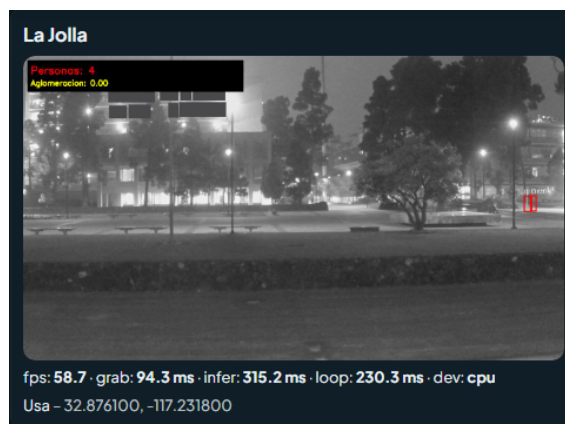
El módulo **GUI** implementa la interfaz de visualización en React.js, responsable de presentar en tiempo real las métricas generadas por el backend, renderizar los streams MJPEG y gestionar la interacción con el operador. Las pruebas unitarias se enfocan en validar el renderizado inicial, la actualización de estados internos, la correcta visualización de flujos de video y el comportamiento de los componentes que representan cámaras individuales.

Para verificar la actualización dinámica de métricas en el componente **CamaraCard**, se simula la recepción de un nuevo conjunto de datos y se comprueba que la vista refleje los cambios:

```
act(() => {
  setMetrics({ count: 8, crowd: 0.21 });
});
expect(screen.getByText(/8 personas/i)).toBeInTheDocument();
```



(a) Componente CamaraCard con métricas antes.



(b) Componente CamaraCard con métricas después (actualizada).

Figura 3.10: Antes/Después de métricas del componente CamaraCard.

También se valida el renderizado correcto del stream procesado para una cámara individual, comprobando que el frontend es capaz de decodificar y mostrar la secuencia MJPEG enviada por el backend sin errores de desincronización:



Figura 3.11: Stream MJPEG renderizado correctamente y con zoom en el componente CamaraCard.

Estas pruebas confirman la fiabilidad del módulo GUI en la representación visual, actualización de datos y consistencia de interacción del usuario con la plataforma.

Pruebas al Módulo ALR (Sistema de Alertas)

El módulo **ALR** gestiona la generación, deduplicación y visualización de alertas mediante notificaciones tipo toast. Además, incorpora mecanismos de polling adaptativo con *backoff* progresivo y

expiración automática de alertas (TTL) después de diez minutos, aun cuando el usuario no las cierre manualmente.

En primer lugar, se fuerza una alerta artificial con valores conocidos para comprobar que el backend la registre:

```
event = {"camera": "La Jolla", "count": 10, "crowd": 0.10}  
alr.push(event)  
assert alr.pending() == 1
```

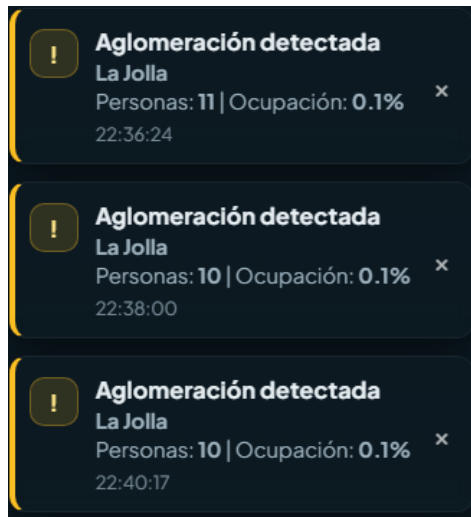


Figura 3.12: Alerta sobre la cámara La Jolla.

Posteriormente, se evalúa la deduplicación combinando el identificador interno y el timestamp del evento, evitando que alertas repetidas se muestren más de una vez. Finalmente, se comprueba la expiración automática (TTL) aplicando un desplazamiento temporal equivalente a diez minutos:

```
jest.advanceTimersByTime(10 * 60 * 1000);  
expect(toastManager.activeCount()).toBe(0);
```

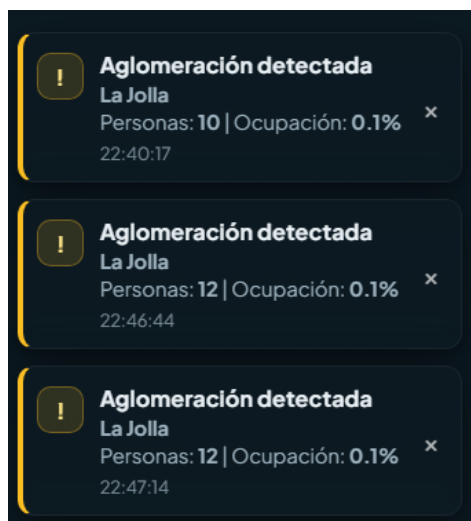


Figura 3.13: Alerta sobre la cámara La Jolla después de 10 minutos.

Como se puede apreciar en las figuras, la alerta con el tiempo 22:36:24 desaparece luego de 10 minutos y la alerta con 22:38:00 se cierra manualmente, comprobando así que el cierre automático y manual. Estas pruebas demuestran que ALR administra correctamente el ciclo de vida de las alertas, evitando acumulaciones injustificadas y garantizando un comportamiento estable frente a eventos recurrentes.

Capítulo 4

Validación

El presente capítulo tiene como objetivo validar la hipótesis planteada en esta memoria, mediante la evaluación sistemática del desempeño del sistema de monitoreo multicámara desarrollado. A diferencia de los capítulos anteriores, centrados en el diseño e implementación de la arquitectura, esta sección se orienta explícitamente a contrastar los objetivos técnicos definidos con evidencia cuantitativa obtenida bajo un protocolo de medición controlado, reproducible y coherente con escenarios de operación reales.

La validación se enfoca en comprobar si el sistema es capaz de operar de manera concurrente y escalable, manteniendo métricas de rendimiento compatibles con aplicaciones de tiempo real desde la perspectiva del operador humano. Para ello, se analizan indicadores de throughput (frames por segundo, FPS), latencia de procesamiento por cuadro (`loop_ms`) y retardo extremo a extremo (cliente-servidor), considerando escenarios progresivos de complejidad creciente que involucran 1, 3 y 6 cámaras simultáneas.

El análisis se centra en el comportamiento del pipeline multicámara, la integración efectiva entre backend y frontend, la calidad de las métricas expuestas al usuario y la robustez del sistema frente a fuentes heterogéneas y condiciones de red variables. Se consideran cámaras IP públicas accesibles mediante RTSP, HTTP y MJPEG, caracterizadas por latencias y bitrates inestables, así como flujos provenientes de YouTube utilizados exclusivamente como referencia comparativa para analizar el comportamiento del sistema bajo condiciones de adquisición más controladas. La inclusión de estas últimas no altera el foco principal del trabajo, que sigue siendo el procesamiento de cámaras IP públicas, pero aporta un punto de contraste útil para interpretar los resultados obtenidos.

Todas las mediciones y evidencias presentadas en este capítulo provienen exclusivamente del **Panel de Validación** implementado en el frontend del sistema, el cual permite instrumentar métricas en tiempo real, visualizar su evolución temporal, contrastarlas con umbrales objetivos y exportar los resultados para su análisis posterior. De esta forma, la validación se apoya en un mecanismo integrado al propio sistema, asegurando coherencia directa entre la arquitectura desplegada y el comportamiento observado.

4.1. Hipótesis y Objetivos de Validación

La hipótesis que guía esta memoria se formula de la siguiente manera:

Es posible implementar un sistema inteligente de monitoreo multicámara capaz de detectar y contabilizar personas en tiempo real, manteniendo un retardo cliente-servidor inferior a

500 ms y un rendimiento operativo estable con al menos cinco cámaras IP simultáneas, ejecutándose en un equipo de especificaciones estándar y sin aceleración por GPU.

A partir de esta hipótesis, el proceso de validación persigue los siguientes objetivos técnicos:

- Verificar que el sistema mantiene un throughput suficiente (FPS) al escalar progresivamente desde una cámara individual hasta seis cámaras simultáneas.
- Evaluar el impacto del escalamiento multicámara sobre la latencia de procesamiento por cuadro (`loop_ms`).
- Medir el retardo extremo a extremo percibido por el operador, considerando la comunicación backend–frontend.
- Analizar la estabilidad del sistema no solo en términos de valores promedio, sino también considerando la variabilidad temporal de las métricas.
- Comprobar que la arquitectura concurrente evita bloqueos globales cuando se incorporan fuentes con latencias de adquisición elevadas.

Para permitir una evaluación objetiva, se definieron los siguientes **criterios cuantitativos esperados**, los cuales se utilizarán posteriormente para clasificar los resultados como satisfactorios o no satisfactorios:

- **FPS por cámara:** $\text{FPS} \geq 10$. Este umbral se adopta como referencia operativa mínima para un sistema de monitoreo visual en tiempo real, donde el objetivo principal es la supervisión de tendencias de afluencia y la detección de eventos relevantes, más que la reproducción fluida de video. En la práctica, cámaras IP públicas y de bajo costo pueden operar a tasas significativamente inferiores (incluso del orden de 2–5 FPS) debido a limitaciones de red, bitrate variable o condiciones externas no controladas. Por tanto, establecer un umbral de 10 FPS permite asegurar una visualización funcional y consistente, manteniendo un margen realista respecto de escenarios operativos adversos.
- **Latencia de procesamiento (`loop_ms`):** $\text{loop_ms} \leq 500$ ms como proxy de latencia por frame.
- **Retardo cliente-servidor:** $\text{retardo} \leq 800$ ms, considerando la diferencia temporal entre el timestamp generado en el backend y su recepción en el frontend.
- **Estabilidad:** evaluación conjunta del promedio y la desviación estándar de cada métrica; cuando resulte pertinente, se utilizará el coeficiente de variación (CV) para caracterizar la dispersión relativa.

Estos valores no representan límites absolutos de diseño, sino referencias operativas que permiten contrastar de forma clara el comportamiento observado del sistema frente a los objetivos planteados.

4.2. Metodología de Validación

4.2.1. Métricas Instrumentadas

El proceso de validación se basa en la recopilación continua de métricas por cámara, las cuales son generadas en el backend y consumidas por el frontend a través de un endpoint dedicado. Las métricas

consideradas son las siguientes:

- **FPS:** número de frames procesados por segundo para cada cámara, calculado a partir del ciclo efectivo de procesamiento.
- **loop_ms:** duración total del ciclo de procesamiento por frame, incluyendo adquisición, inferencia y post-procesamiento.
- **Retardo cliente-servidor (ms):** diferencia temporal entre el timestamp generado en el backend y el momento de recepción en el frontend.

Estas métricas fueron seleccionadas por representar directamente la experiencia del operador: el FPS determina la fluidez visual, la latencia de procesamiento condiciona la capacidad de respuesta del sistema, y el retardo cliente-servidor aproxima el tiempo total que transcurre entre la ocurrencia de un evento en la escena y su visualización en la interfaz.

4.2.2. Restricción de Tiempo Real

En el contexto de esta memoria, el concepto de tiempo real se entiende desde una perspectiva operativa y no estrictamente determinista. Es decir, el sistema se considera apto para operación en tiempo real si:

- el operador recibe información visual y métricas con una latencia suficientemente baja para permitir toma de decisiones,
- la interfaz mantiene fluidez perceptual,
- y las alertas se generan dentro de una ventana temporal razonable respecto al evento observado.

Bajo esta definición, los umbrales establecidos para FPS, `loop_ms` y retardo cliente-servidor constituyen restricciones prácticas que permiten evaluar el cumplimiento del requisito de tiempo real sin exigir garantías temporales rígidas propias de sistemas embebidos críticos.

4.2.3. Protocolo de Medición

Con el fin de garantizar resultados comparables y estadísticamente representativos entre los distintos escenarios evaluados, se definió un protocolo de medición basado en criterios de resolución temporal, estabilidad de las métricas y costo computacional. El intervalo de muestreo se fijó en 3 segundos, ya que este valor permite capturar variaciones operativas relevantes del sistema —como fluctuaciones de carga, latencias de adquisición o eventos de reconexión— sin introducir una sobrecarga innecesaria ni una autocorrelación excesiva entre muestras consecutivas. Este intervalo es además significativamente mayor que la latencia típica por cuadro del pipeline (del orden de cientos de milisegundos), lo que permite que cada muestra represente un promedio efectivo de múltiples frames, reduciendo el ruido instantáneo.

La duración total de cada medición se estableció en 300 segundos (5 minutos) como un compromiso entre cobertura temporal y factibilidad experimental. Este intervalo incluye tanto el periodo inicial de estabilización del sistema como una ventana suficientemente extensa de operación en régimen estacionario, permitiendo observar eventos poco frecuentes sin extender excesivamente el tiempo total

de validación. A partir de esta configuración, el número aproximado de muestras por cámara resulta cercano a 100 (300 s / 3 s), lo que proporciona una base estadística adecuada para estimar promedios, desviaciones estándar y coeficientes de variación con un error estándar reducido, facilitando la comparación objetiva entre los escenarios de 1, 3 y 6 cámaras simultáneas.

Bajo estas consideraciones, la validación se realizó siguiendo un protocolo uniforme para todos los escenarios evaluados, con el fin de asegurar comparabilidad entre los resultados. El protocolo definido es el siguiente:

- **Intervalo de muestreo:** 3 segundos.
- **Duración total de cada medición:** 300 segundos (5 minutos).
- **Número aproximado de muestras por cámara:** 100.

Cada medición se inicia manualmente desde el *Panel de Validación*, una interfaz desarrollada como parte del frontend del sistema, cuya función es visualizar en tiempo real las métricas de rendimiento reportadas por el backend para cada cámara activa. Mediante el control “Iniciar medición”, el panel registra los valores de FPS, latencia de procesamiento (`loop_ms`) y retardo cliente-servidor durante el intervalo definido. Una vez completados los 300 segundos de observación, los gráficos temporales permanecen visibles en pantalla, permitiendo su captura directa para documentación, y los datos asociados a cada métrica pueden exportarse en formato CSV para análisis posterior.

Este diseño metodológico permite delimitar claramente cada ventana de medición, evita ambigüedades temporales entre escenarios y asegura que los resultados reportados correspondan a intervalos de observación homogéneos y reproducibles.

4.2.4. Arquitectura de Medición

El backend del sistema fue desarrollado utilizando FastAPI, donde cada cámara es gestionada por una instancia independiente de `VideoProcessor`, ejecutada en un hilo dedicado. Cada instancia expone sus métricas a través del endpoint:

`/metrics/{cam_id}`

El frontend, implementado en React.js, consume estas métricas y las presenta en el Panel de Validación mediante tarjetas por cámara. Cada tarjeta incluye tres gráficos temporales:

- FPS versus tiempo.
- Latencia de procesamiento (`loop_ms`) versus tiempo.
- Retardo cliente-servidor versus tiempo.

Adicionalmente, cada gráfico incluye una línea de referencia correspondiente al umbral objetivo definido para la métrica, lo que permite una evaluación visual inmediata del cumplimiento o incumplimiento de los criterios establecidos.

4.2.5. Escenarios Evaluados

La validación se estructuró mediante una progresión de escenarios de complejidad creciente:

- **Escenario 1 cámara:** La Jolla.

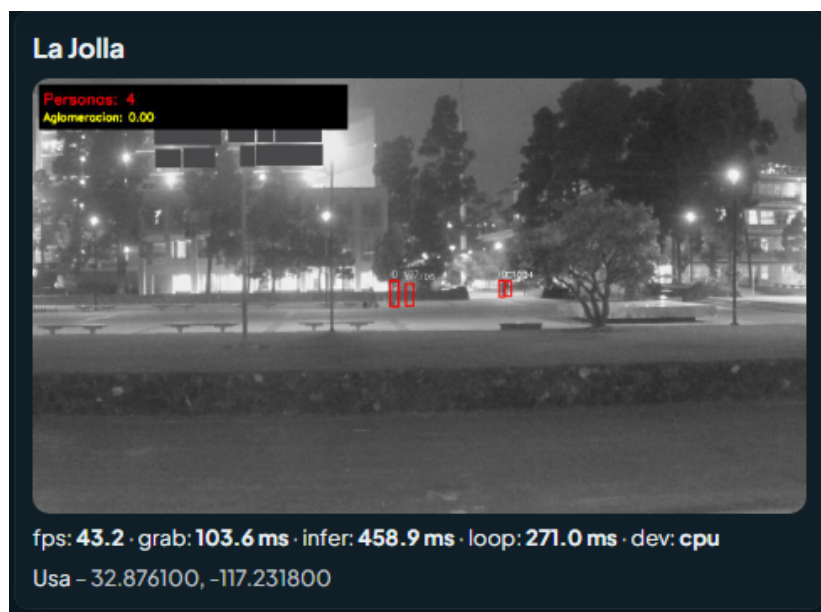


Figura 4.1: Escenario 1 cámara: La Jolla.

- **Escenario 3 cámaras:** La Jolla, Panamá Hospital y Providencia.

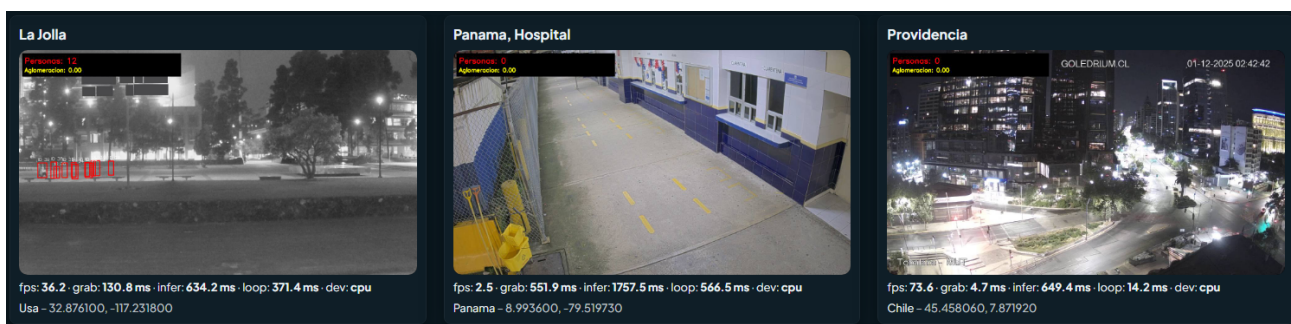


Figura 4.2: Escenario 3 cámaras: La Jolla, Panamá Hospital y Providencia.

- **Escenario 6 cámaras:** La Jolla, Panamá Hospital, Providencia, Piazza Marconi, Buenos Aires y Playa el Quisco.

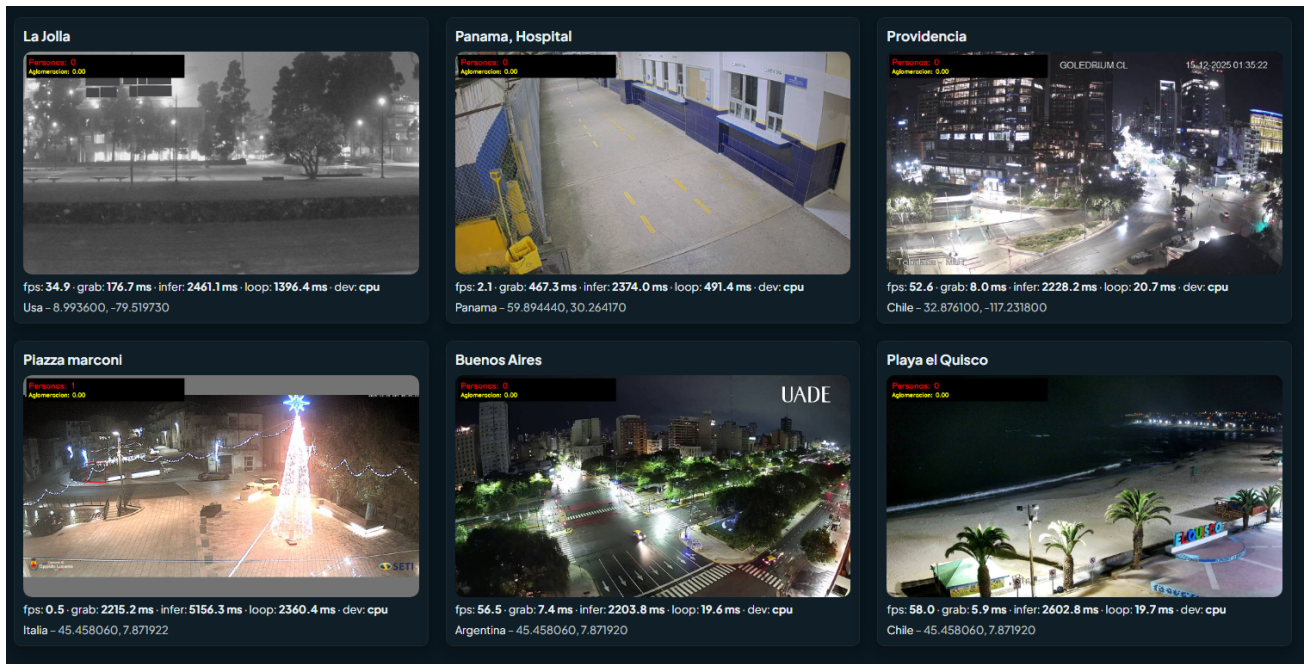


Figura 4.3: Escenario 6 cámaras: La Jolla, Panamá Hospital, Providencia, Piazza Marconi, Buenos Aires y Playa el Quisco.

Esta progresión permitió observar de forma controlada el impacto del escalamiento multicámara sobre las métricas de rendimiento, así como identificar el efecto de fuentes con comportamientos extremos sobre el sistema global.

4.3. Resultados

Los resultados de la validación se presentan organizados por escenario, siguiendo una progresión creciente de complejidad operacional: 1, 3 y 6 cámaras simultáneas. Para cada escenario se reportan métricas cuantitativas obtenidas a partir del Panel de Validación del sistema, incluyendo valores promedio y desviación estándar, junto con una evaluación explícita del cumplimiento de los criterios establecidos para FPS, latencia de procesamiento (`loop_ms`) y retardo cliente-servidor.

Todas las mediciones corresponden a intervalos de observación de 300 segundos, con un muestreo cada 3 segundos, lo que proporciona aproximadamente 100 muestras temporales por cámara y escenario. Esta cantidad de muestras permite caracterizar adecuadamente tanto el valor medio de cada métrica como su variabilidad a lo largo del tiempo.

En los escenarios con múltiples cámaras, se distinguen dos niveles de análisis. Por una parte, se analizan las métricas individuales por cámara, cuya evolución temporal y estabilidad se presentan mediante gráficos. Por otra parte, se reportan métricas agregadas a nivel de escenario, calculadas como el promedio de las métricas individuales de todas las cámaras activas. En estos casos, la desviación estándar agregada corresponde al promedio de las desviaciones estándar individuales de cada cámara, lo que permite caracterizar de manera sintética la variabilidad global del sistema bajo carga concurrente.

Escenario 1 cámara — La Jolla

En el primer escenario se evalúa el comportamiento del sistema operando con una única cámara IP (La Jolla). Este caso base permite establecer una referencia del rendimiento máximo alcanzable bajo carga mínima y condiciones de concurrencia triviales.

Métrica	Media	Desv. Std.	Criterio
FPS	54.0	8.5	OK
loop_ms (ms)	236.3	19.8	OK
Retardo e2e (ms)	11.4	2.9	OK

Cuadro 4.1: Resumen de métricas — Escenario 1 cámara (La Jolla).

Los resultados muestran que, en este escenario, el sistema supera ampliamente los umbrales definidos. El FPS promedio se mantiene significativamente por sobre el mínimo operativo de 10 FPS, mientras que la latencia de procesamiento por frame se mantiene muy por debajo del límite de 500 ms. El retardo cliente-servidor presenta valores bajos y una variabilidad acotada, lo que evidencia una comunicación backend-frontend fluida y estable.

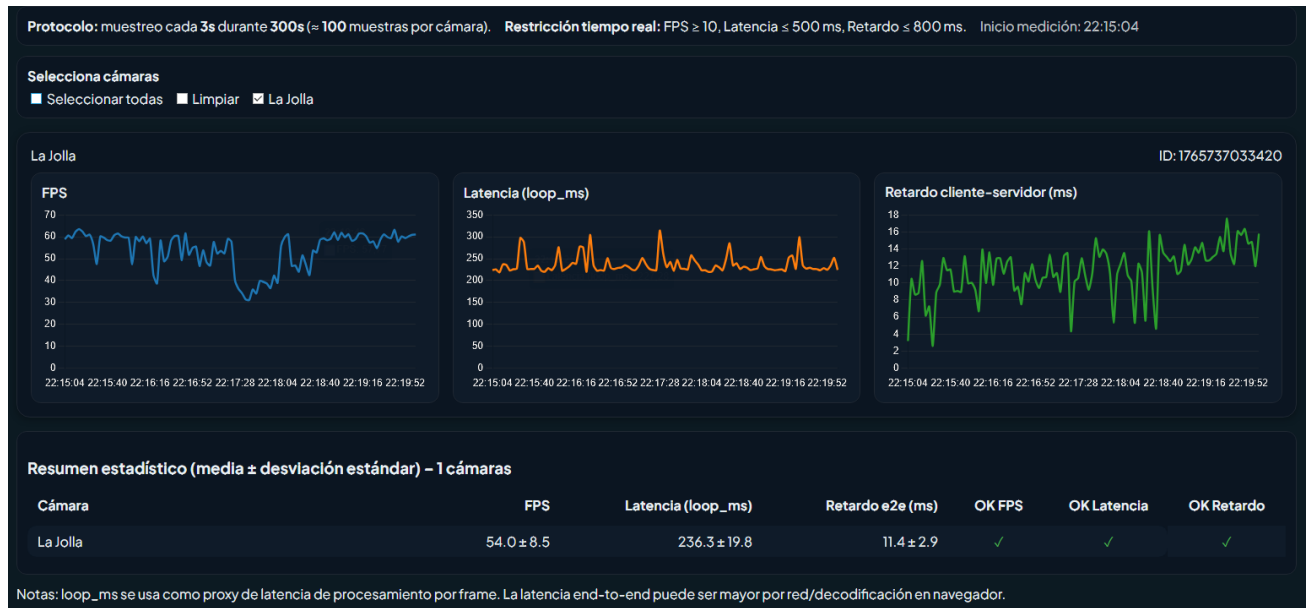


Figura 4.4: Evolución temporal de FPS, latencia y retardo cliente-servidor, junto con resumen estadístico del escenario de 1 cámara.

La estabilidad observada en los gráficos temporales confirma que el sistema es capaz de sostener un rendimiento consistente en condiciones favorables de adquisición, incluso sin aceleración por GPU.

Escenario 3 cámaras — La Jolla, Panamá Hospital y Providencia

El segundo escenario introduce concurrencia real mediante la incorporación de tres cámaras simultáneas, combinando fuentes IP con comportamientos de red heterogéneos. Este escenario permite evaluar el impacto del escalamiento inicial sobre el rendimiento global del sistema y verificar la capacidad de aislamiento entre flujos independientes.

Métrica agregada	Media	Desv. Std.	Criterio
FPS	39.3	5.2	OK
loop_ms (ms)	329.0	44.9	OK
Retardo e2e (ms)	64.7	189.1	OK

Cuadro 4.2: Resumen agregado de métricas — Escenario 3 cámaras.

Las métricas agregadas indican que, a nivel global, el sistema mantiene un desempeño dentro de los umbrales definidos. El FPS promedio se mantiene ampliamente por sobre el mínimo operativo, mientras que la latencia de procesamiento promedio permanece bajo el límite establecido. La desviación estándar refleja la heterogeneidad inherente entre las distintas fuentes de video.

El comportamiento individual de cada cámara se analiza a partir de los gráficos temporales presentados en las figuras siguientes. En particular, la cámara Panamá Hospital exhibe FPS persistentemente bajos y latencias superiores al umbral definido, atribuibles a condiciones adversas de adquisición del flujo IP. En contraste, las cámaras La Jolla y Providencia presentan métricas estables y dentro de los criterios operativos establecidos.



Figura 4.5: Evolución temporal de FPS, latencia y retardo cliente-servidor para las cámaras La Jolla y Panamá Hospital.

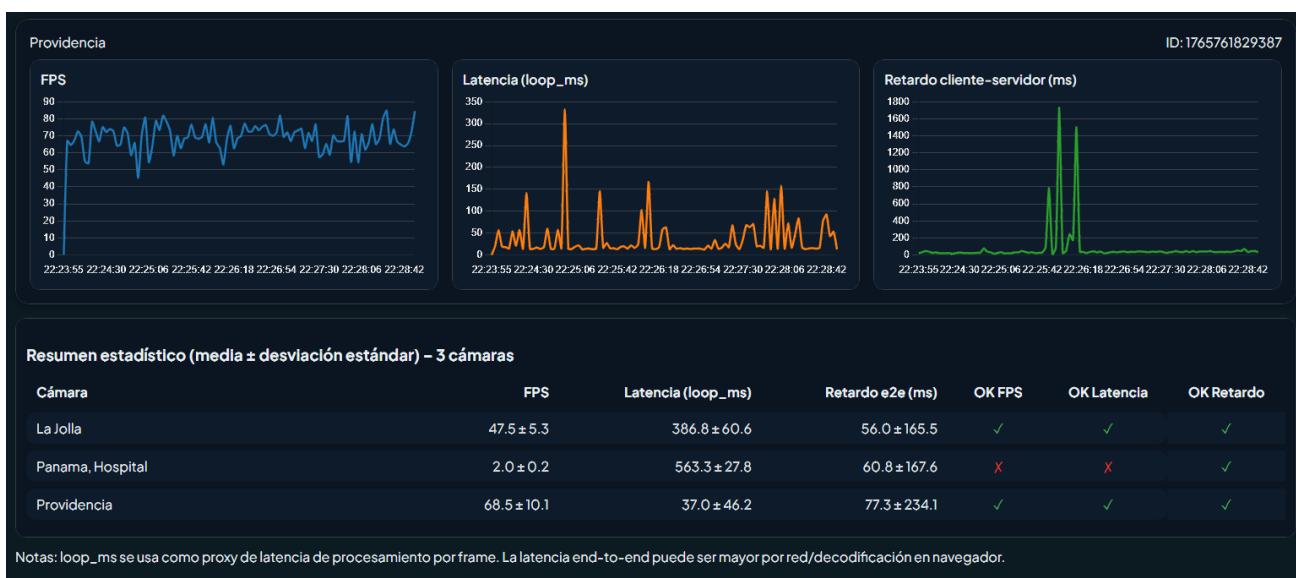


Figura 4.6: Evolución temporal de FPS, latencia y retardo cliente-servidor para la cámara Providencia, junto con resumen estadístico del escenario.

El retardo cliente-servidor presenta una dispersión considerable y la aparición de picos aislados, particularmente en este escenario de concurrencia intermedia. Este comportamiento se explica por la naturaleza de la métrica, la cual mide el desfase temporal entre la marca de tiempo generada en el backend y el instante en que la muestra es recibida y procesada en el navegador cliente. En consecuencia, el valor reportado integra no solo la latencia de red (RTT), sino también efectos de planificación del *event loop* tanto en el servidor como en el cliente, así como retardos asociados a la serialización de datos y al procesamiento del frontend.

Los picos observados corresponden a eventos transitorios y no a una degradación sostenida del sistema. Estos pueden atribuirse a ráfagas puntuales de carga en el backend cuando coinciden tareas de inferencia concurrente, a variabilidad propia de la red (jitter), o a pausas ocasionales en el navegador producto de la gestión de eventos o recolección de memoria. A medida que aumenta el número de cámaras activas, la competencia por recursos de CPU y E/S se incrementa, lo que hace más probable la aparición de estos picos y eleva el valor medio del retardo, sin comprometer la estabilidad global del sistema.

A nivel agregado, los resultados validan el escalamiento inicial del sistema y confirman que el bajo desempeño de una cámara no degrada el funcionamiento de los flujos restantes.

Escenario 6 cámaras — Escenario multicámara completo

El tercer escenario corresponde a la configuración más exigente evaluada, incorporando seis cámaras simultáneas con características de adquisición altamente dispares, tanto en calidad de imagen como en condiciones de red. Este escenario representa el caso límite del sistema y permite evaluar su comportamiento bajo máxima carga concurrente, condición clave para la validación de la hipótesis de escalabilidad planteada en esta memoria.

A diferencia de los escenarios anteriores, el análisis se centra en dos niveles complementarios. En primer lugar, se reportan métricas agregadas a nivel de escenario, las cuales permiten evaluar el cumplimiento global de los criterios de desempeño definidos. En segundo lugar, se analiza el comportamiento

individual de cada cámara a partir de la evolución temporal de las métricas, con el objetivo de identificar fuentes de degradación y su impacto en el rendimiento total.

Métrica agregada	Media	Desv. Std.	Criterio
FPS	36.7	4.0	OK
loop_ms (ms)	615.8	104.9	X
Retardo e2e (ms)	77.1	29.1	OK

Cuadro 4.3: Resumen agregado de métricas — Escenario 6 cámaras.

Las métricas agregadas muestran que el sistema mantiene un FPS medio global ampliamente superior al umbral mínimo operativo, validando la capacidad de procesamiento concurrente incluso bajo carga elevada. Asimismo, el retardo cliente-servidor promedio se mantiene dentro de los límites esperados, lo que indica que la comunicación entre backend y frontend no se ve comprometida por el aumento en el número de flujos.

No obstante, la latencia promedio de procesamiento (`loop_ms`) excede el límite de 500 ms definido como criterio de aceptación. Este resultado no se debe a una degradación homogénea del sistema, sino a la presencia de outliers severos asociados a cámaras específicas con condiciones adversas de adquisición, lo que impacta significativamente el promedio global.

El análisis detallado del comportamiento individual de las cámaras se presenta a través de los gráficos temporales de las Figuras siguientes. Estas visualizaciones permiten observar la evolución de FPS, latencia y retardo cliente-servidor para cada flujo, así como identificar de manera explícita las cámaras que introducen mayor carga al sistema.



Figura 4.7: Evolución temporal de FPS, latencia y retardo cliente-servidor para las cámaras La Jolla y Panamá Hospital.



Figura 4.8: Evolución temporal de FPS, latencia y retardo cliente-servidor para las cámaras Providencia y Piazza Marconi.

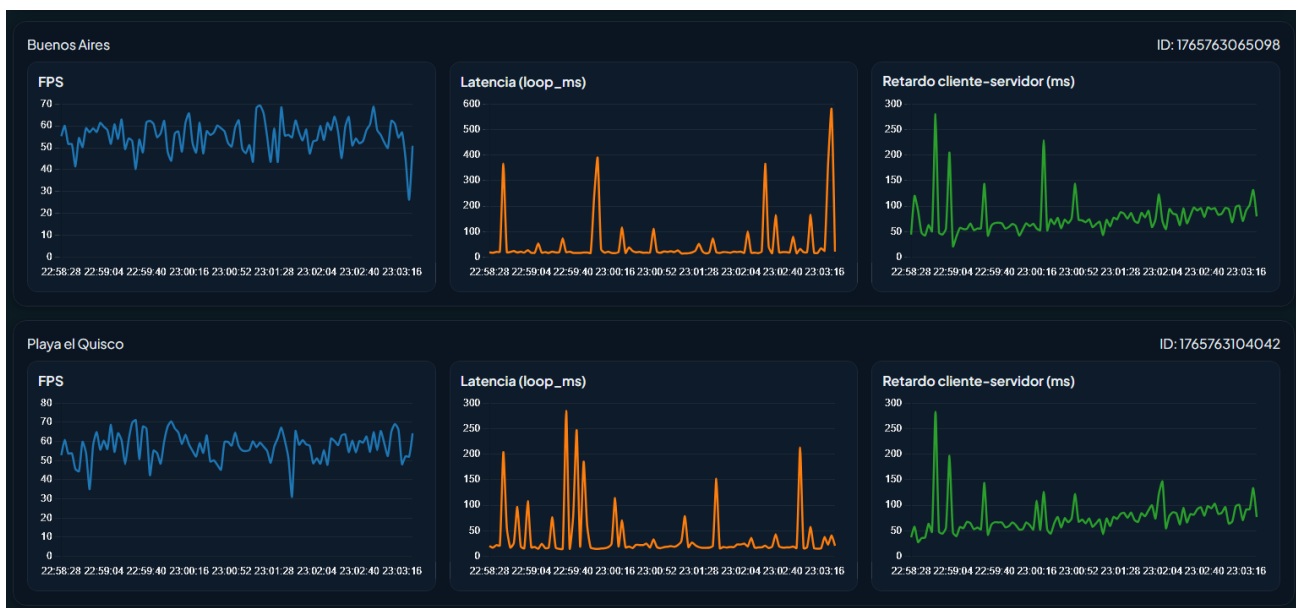


Figura 4.9: Evolución temporal de FPS, latencia y retardo cliente-servidor para las cámaras Buenos Aires y Playa el Quisco.

Cámara	FPS	Latencia (loop_ms)	Retardo e2e (ms)	OK FPS	OK Latencia	OK Retardo
La Jolla	47.0 $\pm$ 2.6	896.4 $\pm$ 58.4	75.2 $\pm$ 21.4	✓	✗	✓
Panama, Hospital	2.1 $\pm$ 0.2	544.7 $\pm$ 94.9	79.2 $\pm$ 32.6	✗	✗	✓
Providencia	58.2 $\pm$ 6.9	27.9 $\pm$ 32.8	76.9 $\pm$ 24.8	✓	✓	✓
Piazza marconi	0.5 $\pm$ 0.1	2137.9 $\pm$ 301.5	76.9 $\pm$ 27.1	✗	✗	✓
Buenos Aires	55.3 $\pm$ 6.8	50.1 $\pm$ 91.9	78.0 $\pm$ 35.7	✓	✓	✓
Playa el Quisco	57.3 $\pm$ 7.2	37.6 $\pm$ 49.9	76.1 $\pm$ 33.1	✓	✓	✓

Notas: loop_ms se usa como proxy de latencia de procesamiento por frame. La latencia end-to-end puede ser mayor por red/decodificación en navegador.

Figura 4.10: Resumen estadístico visual del escenario de 6 cámaras, incluyendo métricas individuales por cámara.

En particular, la cámara Piazza Marconi presenta latencias de procesamiento significativamente superiores al resto, las cuales explican el incumplimiento del criterio de `loop_ms` a nivel agregado. Sin embargo, el análisis temporal confirma que este comportamiento no se propaga al resto de los flujos, evidenciando que el sistema mantiene aislamiento entre cámaras y evita la degradación global del rendimiento.

En conjunto, estos resultados confirman que el sistema es capaz de escalar a múltiples cámaras manteniendo estabilidad operativa en términos de FPS y retardo cliente-servidor. Las principales desviaciones observadas se explican por la heterogeneidad de las fuentes de video y por efectos transitorios asociados a concurrencia y condiciones de red, más que por limitaciones estructurales del diseño, validando así la hipótesis de escalabilidad planteada dentro de los márgenes definidos para esta memoria.

Análisis de Escalabilidad

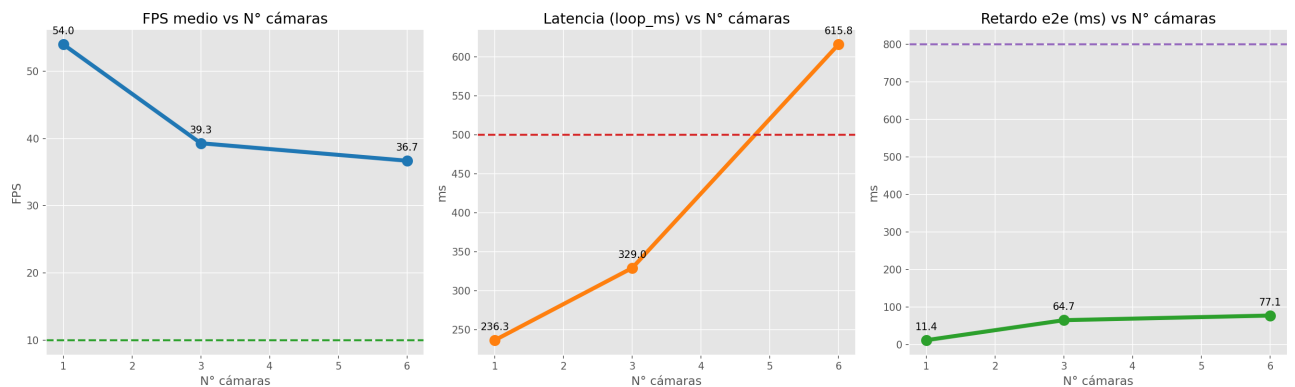


Figura 4.11: Gráficos de FPS medios, latencia media y retardo medio en función del número de cámaras.

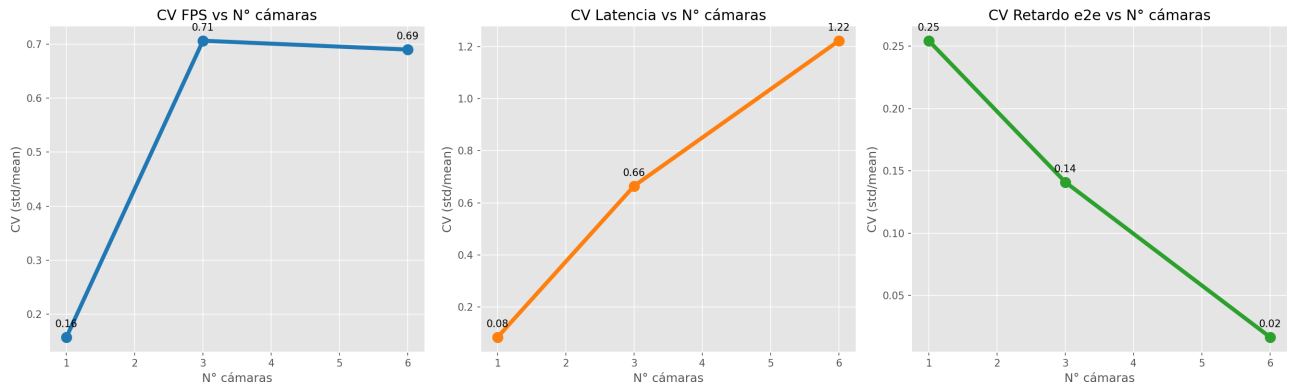


Figura 4.12: Gráficos del coeficiente de variación (CV) de FPS, latencia y retardo en función del número de cámaras.

Las figuras de escalabilidad sintetizan el comportamiento global del sistema al incrementar progresivamente el número de cámaras simultáneas desde uno hasta seis flujos. El análisis conjunto de los valores promedio y del coeficiente de variación (CV) permite evaluar no solo el rendimiento medio del sistema, sino también su estabilidad bajo condiciones de carga creciente.

En términos de throughput, el gráfico de FPS medio evidencia una disminución gradual a medida que aumenta el número de cámaras activas, lo cual es coherente con el incremento en la carga de procesamiento concurrente. No obstante, incluso en el escenario más exigente de seis cámaras, el FPS medio agregado se mantiene ampliamente por sobre el umbral mínimo operativo de 10 FPS, confirmando que la arquitectura multicámara implementada es capaz de escalar sin comprometer la fluidez visual percibida por el operador.

El análisis del coeficiente de variación del FPS revela un comportamiento no lineal al pasar de uno a tres y luego a seis cámaras, con un aumento significativo del CV en el escenario de tres cámaras que se mantiene elevado en el escenario de seis. Este fenómeno no debe interpretarse como una inestabilidad global del sistema, sino como el resultado de una mayor **heterogeneidad entre las fuentes de video**. En particular, al incorporar múltiples cámaras con características de adquisición dispares, se combinan flujos con FPS altos y estables con otros que presentan tasas de cuadros persistentemente bajas, lo que incrementa la dispersión relativa de las métricas agregadas.

Es importante distinguir, en este contexto, entre la **variabilidad intra-cámara**, asociada a la estabilidad temporal de cada flujo individual, y la **variabilidad inter-cámaras**, asociada a la dispersión entre los valores medios de distintas fuentes. Las cámaras con origen en plataformas de streaming como YouTube presentan una cadencia más estable en el tiempo, producto de mecanismos de buffering adaptativo y distribución de contenido, lo que reduce su variabilidad intra-cámara. Sin embargo, la coexistencia de estas fuentes con cámaras IP de baja tasa de cuadros domina el comportamiento del CV agregado, explicando el aumento observado sin implicar una degradación sistémica del sistema.

El comportamiento de la latencia de procesamiento (`loop_ms`) muestra una tendencia creciente conforme se incrementa el número de cámaras, superando el umbral de 500 ms en el escenario de seis flujos. El análisis del CV asociado indica que este incremento no es homogéneo, sino que está fuertemente influenciado por cámaras específicas con condiciones de adquisición adversas. Las fuentes con flujos estables mantienen latencias acotadas y baja dispersión, incluso bajo carga elevada, lo que confirma que el aumento del promedio global está dominado por outliers y no por una degradación generalizada.

Por su parte, el retardo cliente-servidor presenta un comportamiento favorable desde la perspectiva de la hipótesis planteada. A pesar del aumento en la carga concurrente, el retardo medio se mantiene consistentemente bajo y muy por debajo del umbral operativo definido. Adicionalmente, el coeficiente de variación del retardo tiende a disminuir al aumentar el número de cámaras, lo que sugiere un comportamiento más estable y predecible de la comunicación backend–frontend bajo escenarios multicámara.

En conjunto, el análisis de escalabilidad demuestra que el sistema presenta un comportamiento controlado al aumentar el número de flujos simultáneos: el throughput se degrada de forma gradual, la latencia se ve afectada principalmente por fuentes externas con adquisición deficiente y el retardo extremo a extremo permanece estable. Esta evidencia respalda la validez de la arquitectura concurrente propuesta y confirma que el sistema cumple los objetivos operativos definidos para escenarios de monitoreo multicámara en tiempo real.

4.4. Discusión

Los resultados obtenidos a partir del proceso de validación permiten evaluar de manera cuantitativa si el sistema desarrollado cumple con la hipótesis planteada en esta memoria. En particular, los escenarios analizados permiten contrastar el comportamiento del sistema frente a los criterios definidos de throughput, latencia y retardo cliente-servidor, utilizando evidencia numérica y gráfica obtenida directamente desde el Panel de Validación.

En primer lugar, la evidencia recopilada demuestra que la arquitectura basada en hilos independientes logra aislar efectivamente el procesamiento de cada cámara, evitando que la degradación severa de una fuente afecte la continuidad de las demás. Este aspecto constituye uno de los elementos centrales de la hipótesis, ya que la capacidad de procesar múltiples flujos en paralelo sin provocar bloqueos globales es un requisito fundamental para considerar el sistema como escalable. El caso de Piazza Marconi resulta ilustrativo: la adquisición presenta latencias superiores a dos segundos por cuadro, lo que habría paralizado el sistema en una arquitectura secuencial; sin embargo, gracias al diseño concurrente implementado, su impacto se limita exclusivamente a su propio hilo de procesamiento, manteniéndose estables las métricas del resto de las cámaras.

El análisis comparativo entre fuentes IP públicas y flujos provenientes de YouTube permite interpretar adecuadamente los resultados obtenidos. Las cámaras IP, que constituyen el foco principal del proyecto, presentan latencias de adquisición altamente variables debido a congestión de red, bitrate inconsistente y condiciones externas no controladas. Estas variaciones se reflejan en incrementos significativos de `loop_ms` y reducciones abruptas del FPS. En contraste, los flujos provenientes de YouTube ofrecen una adquisición considerablemente más estable, lo que se traduce en valores bajos de latencia y FPS elevados. Esta diferencia confirma que las degradaciones observadas en ciertos escenarios responden al origen del flujo y no a limitaciones estructurales del pipeline desarrollado.

El escalamiento progresivo desde uno hasta seis flujos simultáneos, siendo este último el escenario más exigente evaluado, evidencia una degradación gradual y controlada del rendimiento. El FPS medio del sistema se mantiene siempre por sobre el umbral mínimo definido, validando el criterio de throughput establecido para la hipótesis. En cuanto a la latencia de procesamiento por frame, el criterio de 500 ms se cumple en los escenarios de uno y tres flujos, y se ve comprometido en el escenario de seis cámaras debido a la presencia de outliers severos. No obstante, este incumplimiento no se distribuye

de manera homogénea, sino que se concentra en cámaras con adquisición extremadamente deficiente, mientras que otras fuentes continúan cumpliendo los objetivos definidos.

Un aspecto clave de la validación corresponde al análisis del retardo extremo a extremo. En todos los escenarios evaluados, el retardo cliente-servidor medio se mantiene ampliamente por debajo del umbral de 800 ms definido como criterio operativo. Este resultado confirma que la integración FastAPI-React, junto con el uso de MJPEG y el mecanismo de polling implementado, permite una experiencia de usuario fluida desde la perspectiva del operador humano, incluso bajo carga multicámara.

La inclusión de métricas de desviación estándar y coeficiente de variación permite además caracterizar la estabilidad temporal del sistema más allá de los valores promedio. Cámaras como Providencia, Buenos Aires y Playa el Quisco presentan no solo latencias bajas, sino también una dispersión acotada, lo que indica un comportamiento consistente en el tiempo. En contraste, las cámaras que incumplen los criterios exhiben alta variabilidad temporal, lo que refuerza la interpretación de que los problemas observados están asociados a factores externos de adquisición.

En síntesis, la evidencia cuantitativa y gráfica presentada permite validar la hipótesis planteada en sus aspectos estructurales y de concurrencia, mientras que el criterio de latencia de procesamiento por frame se ve condicionado por la calidad de las fuentes de adquisición. El sistema demuestra capacidad de operación multicámara escalable, mantiene un throughput adecuado y un retardo cliente-servidor dentro de los límites definidos, incluso con seis cámaras simultáneas. La latencia de procesamiento por frame se ve afectada en escenarios extremos debido a fuentes externas, lo que abre oportunidades claras de mejora futura, tales como filtrado de fuentes degradadas, intervalos adaptativos más agresivos y la incorporación de aceleración por GPU. Estas oportunidades son abordadas con mayor profundidad en el capítulo de conclusiones y trabajo futuro.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

En esta memoria se abordó el diseño, implementación y validación de un sistema de monitoreo multicámara orientado a la detección y visualización de afluencia de personas en tiempo real. El objetivo principal consistió en desarrollar una arquitectura capaz de procesar múltiples flujos de video de manera concurrente, garantizando un rendimiento estable, métricas observables y una experiencia de supervisión adecuada para el operador humano.

Desde el punto de vista arquitectónico, se logró implementar un pipeline modular compuesto por los módulos de captura (CAP), procesamiento (PROC), visualización (GUI) y alertas (ALR), integrados mediante una arquitectura backend-frontend basada en FastAPI y React.js. El uso de hilos independientes para cada flujo permitió aislar el procesamiento de las cámaras, evitando bloqueos globales y constituyendo el eje central de la escalabilidad del sistema.

Los resultados de la validación experimental confirmaron que la hipótesis planteada en esta memoria se cumple en sus aspectos fundamentales. En particular, se demostró que el sistema es capaz de operar de forma estable con múltiples cámaras simultáneas, manteniendo un throughput superior al umbral mínimo definido y un retardo cliente-servidor dentro de límites aceptables incluso en escenarios de carga elevada. La degradación del rendimiento observada en ciertos casos se asocia principalmente a condiciones externas de adquisición de los flujos IP y no a limitaciones estructurales del pipeline desarrollado.

En cuanto a la latencia de procesamiento por frame, los escenarios de una y tres cámaras cumplen plenamente los criterios establecidos, mientras que en el escenario más exigente (seis cámaras) se observa un incumplimiento parcial debido a la presencia de fuentes con latencias severas. No obstante, este comportamiento no compromete la estabilidad global del sistema ni la experiencia del operador, ya que las cámaras con condiciones adversas no afectan al procesamiento de las demás.

Respecto a las métricas de alto nivel, el índice de ocupación implementado cumple un rol complementario dentro del sistema, permitiendo visualizar una estimación de densidad en el campo visual. Sin embargo, esta métrica corresponde a un prototipo funcional y no constituye el foco principal de la presente memoria. El autor se limitó a su integración y visualización dentro del sistema, sin profundizar en su optimización ni validación exhaustiva, lo que se refleja en su precisión limitada en ciertos escenarios.

Finalmente, el desarrollo del dashboard multicámara, el sistema de alertas basado en polling adap-

tativo y la generación de reportes permiten afirmar que la solución implementada constituye una plataforma funcional, extensible y adecuada como base para sistemas de monitoreo de afluencia en entornos reales.

5.2. Trabajo Futuro

A partir de los resultados obtenidos y del análisis realizado, se identifican diversas líneas de trabajo futuro que permitirían mejorar y extender la solución desarrollada.

En primer lugar, una mejora natural corresponde a la optimización del índice de ocupación. En su estado actual, esta métrica actúa como un indicador aproximado de densidad visual y presenta limitaciones de precisión, especialmente en escenarios complejos o con oclusiones. Como trabajo futuro, se plantea el desarrollo de métricas de aglomeración más robustas, incorporando modelos espaciales, normalización por perspectiva o técnicas basadas en mapas de densidad, lo que permitiría una estimación más precisa y consistente.

Asimismo, el mecanismo de fallback basado en bounding boxes podría ser reemplazado o complementado con enfoques más avanzados, evitando aproximaciones geométricas que tienden a sobreestimar el área ocupada. Estas mejoras resultarían especialmente relevantes en entornos con alta densidad de personas.

Desde el punto de vista del rendimiento, la incorporación de aceleración por GPU constituye una línea de mejora relevante. El uso de CUDA u otros frameworks de aceleración permitiría reducir significativamente la latencia de procesamiento por frame y mejorar el comportamiento del sistema en escenarios de alta carga multicámara.

Otra línea de trabajo futuro corresponde al refinamiento del sistema de alertas, incorporando políticas más avanzadas de priorización, correlación temporal entre eventos y configuración dinámica de umbrales por cámara. Asimismo, podría integrarse un sistema de persistencia histórica que permita análisis retrospectivos y auditoría de eventos.

Finalmente, el sistema podría extenderse hacia escenarios reales de despliegue, incorporando mecanismos de autenticación más robustos, control de acceso por roles y soporte para almacenamiento persistente de métricas y registros. Estas extensiones permitirían evolucionar la solución desde un prototipo funcional hacia una plataforma de monitoreo de nivel productivo.

En conjunto, estas líneas de trabajo futuro evidencian que la arquitectura desarrollada constituye una base sólida y flexible, sobre la cual es posible construir soluciones más avanzadas para el monitoreo de afluencia de personas en tiempo real.

Bibliografía

- [1] Metro de Santiago. “Metro transportó 57,4 millones de pasajeros durante agosto de 2024”. Comunicado oficial, 5 de septiembre de 2024. Disponible en: <https://www.metro.cl/prensa/noticias/metro-transporto-574-millones-de-pasajeros-durante-agosto-de-2024> [Consultado en octubre de 2025].
- [2] Instituto Nacional de Estadísticas (INE). “Transporte y Comunicaciones: Boletín agosto 2024”. Gobierno de Chile, 2024. Disponible en: <https://www.ine.cl/estadisticas/transporte-y-comunicaciones/boletines> [Consultado en octubre de 2025].
- [3] BNamericas. “Metro de Santiago moves 57.4mn passengers in August 2024”. BNamericas News, 2024. Disponible en: <https://www.bnamericas.com/en/news/metro-de-santiago-moves-574mn-passengers-in-august-2024> [Consultado en octubre de 2025].
- [4] Metro de Santiago. “Informe de gestión 2024”. Departamento de Planificación y Desarrollo Operacional, Metro S.A., 2024. Disponible en: <https://www.metro.cl/documentos/informes-gestion> [Consultado en octubre de 2025].
- [5] Blake Blackshear. *Frigate* — Network Video Recorder with realtime object detection. GitHub repository. 2019. <https://github.com/blakeblackshear/frigate>.
- [6] R. M. B. S. Wicaksono, I. W. Mustika, S. Sulisty. “Region-Based People Counting with Embedded System in Smart Building Environment”. IEEE (2024). (PDF disponible en el repositorio de archivos adjuntos).
- [7] Taekwon Chung, Seunghyeon Park, Sanghun Kim, Kiwoong Kwon. “Minimizing FPS Degradation in Multi-Streaming Processing with YOLO”. IEEE COMPSAC 2024, DOI:10.1109/COMPSAC61105.2024.00227.
- [8] (Artículo de revisión / caso práctico) Varios autores. Ejemplos de estudios en *Sensors* y publicaciones relacionadas con conteo y monitoreo de personas (por ejemplo: <https://www.mdpi.com/1424-8220/21/9/2958>). [consultado por el autor].
- [9] Preprint reciente (arXiv:2401.14132). Artículos recientes que proponen optimizaciones y arquitecturas orientadas a multi-camera real-time detection. <https://arxiv.org/abs/2401.14132>.
- [10] Axis Communications. “Axis Camera Station — Video Management Software.” Documentación oficial del producto, Axis Communications AB, 2024. Disponible en: <https://www.axis.com/products/axis-camera-station>.

- [11] Milestone Systems. “Milestone XProtect — Video Management Software.” Documentación oficial, 2024. Disponible en: <https://www.milestonesys.com/es/products/software/xprotect>
- [12] Motorola Solutions. “Avigilon Control Center (ACC) — Intelligent Video Management Software.” Motorola Solutions Inc., 2024. Disponible en: https://www.motorolasolutions.com/es_xl/video-security-analytics/avigilon-control-center.html
- [13] Motorola Solutions. “Avigilon Control Center 7 — Hoja de datos técnica (v7.12).” Documento técnico oficial, Motorola Solutions Inc., 2024. Disponible en: https://d8eqw8u9b6kgn.cloudfront.net/file_library/pdf/acc7/ACC_712_datasheet_ES-LA.pdf
- [14] J. Redmon et al. “You Only Look Once: Unified, real-time object detection”. CVPR 2016.
- [15] Ultralytics. “YOLO — A brief history.” Documentación oficial, 2024. Disponible en: <https://docs.ultralytics.com/#yolo-a-brief-history>.
- [16] A. Bewley, Z. Ge, L. Ott, F. Ramos, B. Upcroft. “Simple Online and Realtime Tracking”. 2016.
- [17] N. Wojke, A. Bewley, D. Paulus. “Simple online and realtime tracking with a deep association metric”. ICIIP 2017.
- [18] FFmpeg developers. FFmpeg multimedia framework. <https://ffmpeg.org>.
- [19] NVIDIA DeepStream SDK — documentación técnica. <https://developer.nvidia.com/deepstream-sdk>.