UNIVERSIDAD INTERNACIONAL DEL ECUADOR

Powered by Arizona State University

FACULTAD DE INGENIERIAS DIGITALES Y TECNOLOGIAS EMERGENTES

CARRERA DE INGENIERIA EN TECNOLOGIAS DE LA INFORMACION

Evaluación del tiempo de respuesta de la cadena de notificación de un sistema de detección de armas basado en inteligencia artificial para espacios públicos comerciales.

Trabajo de Integración Curricular previo a la obtención del título de:

INGENIERO EN TECNOLOGIAS DE LA INFORMACION

Autor: Chicaiza Rodriguez Lander Andres

Tutor: Valarezo Leon Dario Javier.

LOJA

2026

Aprobación del Tutor del Trabajo de Integración Curricular

Yo Darío Javier Valarezo León, en calidad de tutor del Trabajo de Integración Curricular denominado: “Evaluación del tiempo de respuesta de la cadena de notificación de un sistema de detección de armas basado en inteligencia artificial para espacios públicos comerciales”, realizado por Lander Andrés Chicaiza Rodríguez ha sido orientado y revisado en todas sus partes y considero que cumple los requisitos establecidos por la institución. En virtud de ello doy mi aprobación a fin de continuar con el proceso académico correspondiente.

Mgs. Darío Javier Valarezo León

Tutor del Trabajo de Integración Curricular

Declaración de autoría y cesión de derechos

Yo, Lander Andrés Chicaiza Rodríguez declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido presentado anteriormente para ningún grado o calificación profesional y que se ha consultado la bibliografía detallada.

Cedo mis derechos de propiedad intelectual a la Universidad Internacional del Ecuador, para que sea publicado y divulgado en internet, según lo establecido en la Ley de Propiedad Intelectual, Reglamento y Leyes**.**

_____

Lander Andrés Chicaiza Rodríguez

Dedicatoria

A mi familia, pilar fundamental de mi vida y fuente inagotable de apoyo. Su amor incondicional y apoyo siempre incondicional han sido la fuerza que me ha ayudado a superar cada obstáculo y siempre alcanzar mis metas.

A mis padres, por su esfuerzo y por enseñarme el valor del esfuerzo y sacrificio. Gracias por sus sabias crianzas han hecho de mi lo que soy hoy en día.

Y finalmente, a mi lindo Ecuador, cuya realidad actual han inspirado esta investigación. Espero que este trabajo realizado contribuya, aunque sea en una mínima o pequeña medida que sea este inicio para cambiar la situación actual, a partir de más sistemas más especializados para un país más seguro para todos los ecuatorianos.

Agradecimiento

Al concluir esta etapa de mi formación académica, deseo expresar mi más sincero agradecimiento a aquellas personas que han sido fundamentales durante este gran proceso.

En primer lugar, a mis padres, por su amor infinito, su fe y su confianza en mí. Sus sacrificios no han sido en vano, esta es la razón por la que hoy estoy aquí.

A mis abuelos, Bolívar y Marianita, por ser mis segundos padres durante toda mi vida. Su sabiduría y amor incondicional han sido el pilar fundamental de lo que me he convertido hoy.

A mi tutor, Darío, por su invaluable guía, paciencia, dedicación y predisposición para ayudarme. Sus conocimientos y consejos han sido pilares esenciales en la construcción de este proyecto.

Y un agradecimiento especial para Lorena Conde, que siempre ve lo mejor para todos los estudiantes como directora, siempre busca que todos sean unidos para ella una felicitación y agradecimiento que siempre fue como una segunda madre para todos los estudiantes de la carrera.

Índice de contenido

Aprobación del Tutor del Trabajo de Integración Curricular II

Declaración de autoría y cesión de derechos III

Dedicatoria IV

Agradecimiento V

Índice de contenido VI

Índice de tablas XI

Índice de figuras XIII

Lista de acrónimos y abreviaturas XV

Resumen 19

Abstract 20

Introducción 21

Planteamiento del problema 22

Objetivos: 24

Objetivo General 24

Objetivos Específicos: 24

Pregunta de Investigación: 25

Justificación 25

Alcance y limitaciones 26

Capítulo I 28

Estado del Arte 29

1.1. Inteligencia Artificial y Visión por Computadora..... 29

1.1.1. Fundamento de la inteligencia artificial. 29

1.1.2. ¿Qué es la visión por computadora? 30

1.1.3. Revolución de la visión por computadora a través de la IA. 31

1.2. Técnicas de IA en Visión por Computadora..... 32

1.2.1. Redes Neuronales Convolucionales (CNN): estructura y aprendizaje..... 32

1.2.2. Modelos R-CNN, Fast R-CNN, YOLO: evolución y principios 35

1.2.3. Procesamiento de imágenes y video para detección de objetos. 37

1.3. Modelos YOLO y su Aplicación en Tiempo Real..... 39

1.3.1. Principios y arquitectura de YOLO. 39

1.3.2. Avances en las versiones YOLOv5 y YOLOv8..... 40

1.3.3. Ventajas de YOLO para detección en tiempo real 41

1.4. Aplicaciones de IA en Sistemas de Seguridad y Vigilancia..... 41

1.4.1. Casos de éxito en videovigilancia basada en IA 41

1.4.2. Algoritmos y plataformas para monitoreo inteligente..... 42

1.4.3. Desafíos actuales: privacidad y robustez..... 43

1.5. Detección de Armas Mediante Inteligencia Artificial 44

1.5.1. Complejidades técnicas: variedad, oclusión, entorno..... 44

1.5.2. Datasets específicos y técnicas de aumento de datos..... 45

1.5.3. Transfer learning y fine-tuning en modelos de armas..... 47

1.5.4. Métricas de evaluación: precisión, recall, mAP, F1-score 47

1.5.5. Validación en escenarios reales y selección de umbrales..... 48

1.5.6. El tiempo de respuesta en los sistemas de alerta: lo que se mide y lo que no 49

1.6. Consideraciones Éticas y Legales en la Implementación 51

1.6.1. Protección de datos y privacidad en vigilancia digital 51

1.6.2. Prevención de sesgos algorítmicos en seguridad pública 51

1.6.3. Normativas y marco regulatorio nacional e internacional 52

Capítulo II 54

2. Análisis y Diseño 55

2.1. Introducción al análisis del problema 55

2.1.1. Motivaciones prácticas para el desarrollo del sistema 55

2.1.2. Contexto de aplicación en seguridad ciudadana 55

2.2. Metodología empleada: CRISP-DM 56

2.2.1. Justificación del uso de CRISP-DM en proyectos de IA 56

2.2.2. Fases del modelo aplicadas al sistema de detección 57

2.3. Levantamiento de requerimientos 57

2.3.1. Fuentes informales de recolección (casos reales) 58

2.4. Requisitos del sistema 58

2.4.1. Requisitos funcionales y no funcionales 58

2.4.2. Interfaces del sistema (usuario, hardware, software, comunicación) 60

2.4.3. Reglas de negocio y condiciones de operación 60

2.5. Modelo de casos de uso 61

2.5.1. Descripción de actores y funcionalidades principales 61

2.5.2. Diagramas UML de casos de uso generales y específicos 62

2.6. Diagramas de análisis 69

2.6.1. Diagrama de clases principal 69

2.6.2. Diagrama de secuencia: proceso de detección y alerta 71

2.6.3. Diagrama de estados: ciclo de vida de una alerta 74

2.7. Diseño de la arquitectura del sistema 76

2.7.1. Arquitectura cliente-servidor y flujo de información 77

2.7.2. Estructura del backend (API REST, Django ORM) 81

2.7.3. Diseño del frontend: PyQt5 y web dashboard 85

Capítulo III 88

3. Desarrollo e Implementación 89

3.1. Entorno de desarrollo y herramientas tecnológicas 89

3.1.1. Lenguajes, frameworks y bibliotecas utilizadas 89

3.1.2. Justificación técnica de la pila de desarrollo 91

3.2. Implementación del modelo de IA 97

3.2.1. Selección y entrenamiento de YOLOv8 97

3.2.2. Ajuste fino (fine-tuning) y validación del modelo 100

3.2.3. Integración con flujo de captura en tiempo real 104

3.3. Desarrollo de la interfaz cliente (PyQt5) 107

3.3.1. Visualización de video y detecciones 107

3.3.2. Controles y envío de alertas 110

3.4. Desarrollo del backend y API REST (Django) 115

3.4.1. Almacenamiento de alertas e imágenes 115

3.4.2. Gestión de usuarios y autenticación 117

3.4.3. Envío de notificaciones por correo electrónico 120

3.5. Desarrollo del panel web de visualización 122

3.5.1. Dashboard de alertas en tiempo real 122

3.5.2. Filtros de búsqueda y detalles individuales 125

3.6. Validación del sistema 129

3.6.1. Escenarios de prueba y resultados funcionales 129

3.6.3. Pruebas de rendimiento: tiempo de detección, envío y consulta 133

3.7. Validación con usuarios: retroalimentación recibida 138

Capítulo IV 144

Conclusiones y Recomendaciones 145

Conclusión general: 145

Conclusiones según objetivos específicos: 145

Recomendaciones 147

Referencias 149

Índice de tablas

Tabla 1: *Comparación entre visión por computadora tradicional y visión por computadora basada en IA 32*

Tabla 2: *Componentes fundamentales de una red neuronal convolucional 34*

Tabla 3: *Comparación entre modelos R-CNN, Fast R-CNN, Faster R-CNN y YOLO 36*

Tabla 4: *Técnicas comunes de procesamiento aplicadas a imágenes y video para visión por computadora..... 38*

Tabla 5: *Comparación técnica entre YOLOv5 y YOLOv8 ..... 40*

Tabla 6: *Principales desafíos técnicos en la detección de armas mediante IA 45*

Tabla 7: *Datasets relevantes para detección de armas con IA 46*

Tabla 8: *Métricas de evaluación para modelos de detección de armas. 48*

Tabla 9 *Alcance temporal de las mediciones reportadas en los trabajos revisados 50*

Tabla 10: *Principales marcos normativos aplicables a la vigilancia con IA 53*

Tabla 11: *Aplicación de las fases CRISP-DM al sistema propuesto 57*

Tabla 12: *Requisitos funcionales y no funcionales del sistema. 59*

Tabla 13: *Interfaces del sistema..... 60*

Tabla 14: *Actores principales del sistema y sus funcionalidades..... 62*

Tabla 15 *Herramientas tecnológicas utilizadas en el desarrollo del sistema..... 89*

Tabla 16: *Criterios de decisión. 96*

Tabla 17 *Hiperparámetros de entrenamiento 99*

Tabla 18 *Técnicas de aumento de datos..... 100*

Tabla 19 *Métricas finales del modelo YOLOv8 tras fine-tuning..... 103*

Tabla 20 *Detalle acerca de controles del sistema de monitoreo 111*

Tabla 21 *Campos del modelo UploadAlert 115*

Tabla 22 *Campos de búsqueda disponibles en el sistema. 126*

Tabla 23 *Matriz de las pruebas funcionales ejecutadas..... 130*

Tabla 24 *Resultados de rendimiento del sistema. 134*

Tabla 25 *Distribución percentil de tiempos de respuesta..... 135*

Tabla 26 *Operación de sistema bajo 100 ciclos..... 136*

Tabla 27 *Perfiles de usuarios participantes 139*

Tabla 28 *Evaluación Consolidada 141*

Índice de figuras

Figura 1: *Evolución del total de homicidios y tasa por cada 100.000 habitantes en Ecuador (2019-2023)* *22*

Figura 2: *Evolución del desarrollo de la IA, desde los enfoques simbólicos hacia el aprendizaje automático y profundo.* *30*

Figura 3: *Arquitectura básica de una red neuronal convolucional (CNN): entrada, extracción de características, clasificación y salida probabilística.* *34*

Figura 4: *Arquitecturas de detección de objetos: Faster R-CNN, SSD (Single Shot Detector), CenterNet y YOLO.* *37*

Figura 5: *Detección de objetos con YOLO* *39*

Figura 6: *Arquitectura general de un sistema de videovigilancia inteligente basado en IA* *43*

Figura 7: *Diagrama general de casos de uso del sistema de detección de armas* *64*

Figura 8: *Diagrama de casos de uso - Módulo de Autenticación.* *66*

Figura 9: *Casos de uso, Modulo de Detección y Alerta.* *67*

Figura 10: *Diagrama de casos de uso - Módulo de Consulta y Visualización.* *68*

Figura 11: *Diagrama de clases del sistema de detección de armas con arquitectura cliente-servidor* *70*

Figura 12: *Diagrama de secuencia del proceso de detección y alerta en tiempo real.* *73*

Figura 13: *Diagrama de estados del ciclo de vida de una alerta de detección de armas.* . *76*

Figura 14: *Arquitectura cliente-servidor del sistema de detección de armas* *78*

Figura 15: *Flujo de información del sistema de detección de armas* *80*

Figura 16: *Modelo entidad-relación (ERD) del sistema backend* *82*

Figura 17: *Endpoints del servidor Django para autenticación, gestión de alertas y navegación web* *84*

Figura 18: *Esquema de navegación - aplicación de escritorio (PyQt5). 86*

Figura 19: *Esquema de navegación del dashboard web del sistema 87*

Figura 20 *Configuración archivo data.yaml 98*

Figura 21 *Evolución de Loss Functions 101*

Figura 22 *Métricas de Precisión y Recall 102*

Figura 23 *Evolución de mAP 102*

Figura 24 *Learning Rate Schedule 103*

Figura 25 *Diagrama de flujo en tiempo real 104*

Figura 26 *Estructura de datos enviados 106*

Figura 27 *Inicialización de CameraLabel en detectionWindowDual.py 108*

Figura 28 *Interfaz de monitoreo mostrando el stream en vivo y recursos del sistema 109*

Figura 29 *Interfaz gráfica del sistema de detección 113*

Figura 30. *Generación automática de tokens con post_save 118*

Figura 31 *Configuración del endpoint de autenticación 118*

Figura 32. *Formularios gráficos de inicio de sesión y registro. 119*

Figura 33 *Interfaz Aplicación de fechas y filtros 123*

Figura 34 *Aplicación de filtro activado 124*

Figura 35 *Interfaz paginación en el panel de alertas 124*

Figura 36 *Aplicación de filtros por diferentes tipos de búsqueda: ubicación, receptor y fechas 127*

Figura 37 *Visualización de detalles individuales de una alerta 128*

Lista de acrónimos y abreviaturas

API: Application Programming Interface (Interfaz de Programación de Aplicaciones)

CNN: Convolutional Neural Networks (Redes Neuronales Convolucionales)

CPU: Central Processing Unit (Unidad Central de Procesamiento)

CRISP-DM: Cross Industry Standard Process for Data Mining (Proceso Estándar de la Industria para la Minería de Datos)

GPU: Graphics Processing Unit (Unidad de Procesamiento Gráfico)

GUI: Graphical User Interface (Interfaz Gráfica de Usuario)

HTTPS: Hypertext Transfer Protocol Secure (Protocolo Seguro de Transferencia de Hipertexto)

IA: Inteligencia Artificial

INEC: Instituto Nacional de Estadística y Censos

IoU: Intersection over Union (Intersección sobre Unión)

RTSP: Real Time Streaming Protocol (Protocolo de Transmisión en Tiempo Real)

SMTP: Simple Mail Transfer Protocol (Protocolo Simple de Transferencia de Correo)

SSD: Single Shot Detector (Detector de Disparo Único)

UI: User Interface (Interfaz de Usuario)

YOLO: You Only Look Once (Solo Miras Una Vez)

Resumen

Este trabajo construye un sistema de detección de armas sobre YOLOv8 y lo usa como instrumento de medición: lo que se mide es el tiempo que separa la aparición del arma ante la cámara de la llegada del aviso al personal del local. El entrenamiento empleó 4.000 imágenes a lo largo de 800 épocas. La arquitectura es cliente-servidor: un cliente PyQt5 lee cámaras IP por RTSP e infiere en la propia máquina, un backend en Django persiste las alertas en PostgreSQL y el correo sale por la API HTTP de Brevo. Siguiendo CRISP-DM, el autor partió el tiempo de respuesta en cuatro tramos, detección (T1), envío al servidor (T2), despacho (T3) y entrega al receptor (T4), y los midió sobre 100 ciclos controlados en dos comercios de Loja con umbral de 0,75. El aviso tardó 5.559,8 ms de promedio, bajo la meta de siete segundos. De ese total, la entrega del correo absorbió el 57,3 %; la inferencia local, el 1,3 %. El modelo trabajó en campo con 82,4 % de precisión y 84,0 % de recall.

Palabras clave: cadena de notificación, detección de armas, latencia extremo a extremo.

Abstract

This work builds a weapon detection system on YOLOv8 and uses it as a measuring instrument: what gets measured is the time between the weapon appearing on camera and the alert reaching the people working in the shop. Training used 4,000 labeled images across 800 epochs. The architecture is client-server: a PyQt5 desktop client reads IP cameras over RTSP and infers locally, a Django backend persists alerts on PostgreSQL, and email goes out through the Brevo HTTP API. Following CRISP-DM, the author split response time into four segments, detection (T1), transmission to the server (T2), notification dispatch (T3) and delivery to the recipient (T4), then measured them over 100 controlled cycles in two shops in Loja, Ecuador, at a 0.75 confidence threshold. The alert averaged 5,559.8 ms, below the seven-second target. Of that total, email delivery absorbed 57.3%; local inference, 1.3%. In the field the model worked at 82.4% precision and 84.0% recall.

Keywords: notification chain, weapon detection, end-to-end latency.

Introducción

Ecuador cerró 2023 con 8.004 homicidios. En 2019 habían sido 1.187 (Observatorio Ecuatoriano de Crimen Organizado, 2024)Sobre ese fondo opera el comercio urbano de pequeña escala: locales que ya pagaron una o dos cámaras y que graban el asalto con nitidez suficiente para denunciarlo. Para denunciar sirven. Para reaccionar mientras el asalto ocurre, no.

Un modelo de visión por computadora sí avisa mientras ocurre. Los detectores de una sola pasada, entre ellos la familia You Only Look Once (YOLO), reconocen objetos en decenas de milisegundos sobre hardware modesto. Eso basta para analizar en vivo el flujo de una cámara IP. El Capítulo I recorre esa evolución, desde las arquitecturas de dos etapas hasta YOLOv8, y revisa los conjuntos de datos, las métricas de evaluación y las restricciones éticas y legales que pesan sobre la videovigilancia automatizada en Ecuador.

Este trabajo levanta un sistema completo sobre esa base. El cliente de escritorio, escrito en PyQt5, consume las cámaras del local mediante Real Time Streaming Protocol (RTSP) y procesa cada cuadro en la propia máquina, sin sacar el video del establecimiento. Cuando el modelo supera 0,75 de confianza, ese mismo cliente sube la alerta y su fotografía a un backend en Django respaldado por PostgreSQL. El backend dispara entonces un correo por la interfaz de programación de aplicaciones (API) HTTP (Hypertext Transfer Protocol) de Brevo hacia el contacto que cada local designe. El sistema se construyó para medir sobre él. Ahí está la diferencia con los trabajos revisados: se detienen en el detector.

El autor midió esa cadena entera, tramo por tramo (T1 a T4), durante doce días seguidos en dos comercios del centro de Loja. El resto del documento sigue ese recorrido: el Capítulo II se ocupa del análisis y el diseño; el Capítulo III, de la implementación y de las pruebas de rendimiento con los tiempos de cada tramo; el

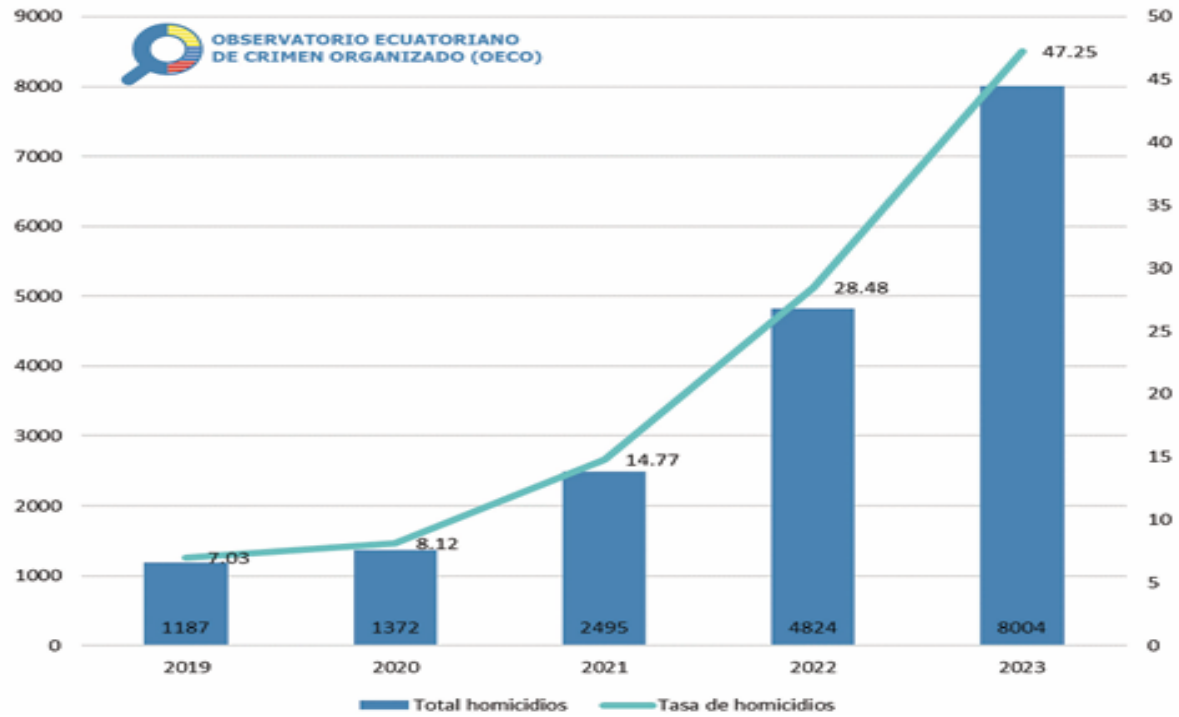
Capítulo IV cierra con las conclusiones por objetivo y las recomendaciones para quien quiera desplegar algo semejante.

Planteamiento del problema

Ecuador atraviesa el deterioro de seguridad más agudo de su historia reciente. Entre 2019 y 2023 los homicidios pasaron de 1.187 a 8.004, un aumento del 574 %, y la tasa por cada 100.000 habitantes subió de 7,03 a 47,25 (Observatorio Ecuatoriano de Crimen Organizado, 2024) La Figura 1 recoge ambas curvas. El dato sale del registro oficial de muertes violentas y no de encuestas de victimización, de modo que mide hechos consumados antes que percepciones.

Figura 1:

Evolución del total de homicidios y tasa por cada 100.000 habitantes en Ecuador (2019-2023)



Nota. Adaptado de (Observatorio Ecuatoriano de Crimen Organizado, 2024).

Detrás de esa curva hay varias causas superpuestas. La pandemia por COVID-19 dejó secuelas socioeconómicas que empujaron a parte de la población hacia

economías ilegales; en paralelo, organizaciones criminales transnacionales se expandieron y el narcotráfico penetró zonas urbanas y rurales antes ajenas al negocio. A eso se sumó una crisis penitenciaria que dejó al Estado sin control sobre los centros de rehabilitación social. El salto más brusco ocurrió entre 2020 y 2021: la tasa pasó de 7,82 a 14,12 por cada 100.000 habitantes, un aumento cercano al 80 % en un solo año (Banco Mundial, 2024).

El homicidio, con todo, no es lo que el comerciante enfrenta a diario. El robo agravado y la extorsión crecieron en paralelo, y golpean directo al local pequeño: las denuncias por extorsión pasaron de 862 en 2019 a 21.811 en 2023, y tres de cada diez empresas del país fueron víctimas de ese delito durante 2023, la mayoría más de una vez (Observatorio Ecuatoriano de Crimen Organizado, 2024). Siete de cada diez afectadas ni siquiera denunciaron. Frente a ese cuadro, el modelo de vigilancia vigente responde tarde: las cámaras documentan el hecho para la investigación posterior, pero rara vez alguien las mira mientras ocurre.

La videovigilancia asistida por computador cambia ese papel. Un detector que reconoce el arma y dispara el aviso sin que nadie mire la pantalla convierte la cámara en un sensor activo, y unos segundos bastan para que el personal pulse el botón de pánico o se resguarde. Este trabajo construye un sistema de ese tipo para espacios comerciales, dentro de las restricciones de la LOPDP que detalla el Capítulo I. Su problema de investigación está en un punto ciego: los trabajos revisados reportan la inferencia en milisegundos y sobre esa cifra prometen tiempo real. Pero la inferencia es solo el primer eslabón. Le siguen el envío de la evidencia al servidor, su procesamiento, el despacho de la notificación y la entrega al destinatario, tramos que dependen de la red del local y de proveedores externos. Quien no descompone ese trayecto optimiza a ciegas: afina el modelo cuando el retraso estaba en el ancho de banda.

Objetivos:

Objetivo General

Evaluar el tiempo de respuesta de la cadena de notificación de un sistema de detección de armas basado en inteligencia artificial para espacios públicos comerciales.

Objetivos Específicos:

- Diseñar un protocolo de validación replicable que establezca las condiciones mínimas de operación del sistema de detección de armas basado en inteligencia artificial y el procedimiento de instrumentación de marcas de tiempo en los establecimientos comerciales participantes.
- Validar el comportamiento del modelo de detección de armas en espacios públicos comerciales, contrastando su precisión, recall y F1-score en operación real frente a los valores obtenidos en la fase de entrenamiento.
- Caracterizar la cadena de notificación en cada uno de sus eslabones, medidos como los intervalos entre la captura del cuadro y la detección positiva (T1), entre la detección positiva y su recepción en el servidor (T2), entre la recepción en el servidor y el despacho de la notificación (T3) y entre el despacho y la llegada de la alerta al dispositivo del personal del establecimiento (T4).
- Identificar los puntos de retraso recurrentes en la cadena de notificación con el fin de formular recomendaciones técnicas y operacionales para optimizar el sistema.

Pregunta de Investigación:

¿Cómo se distribuye el tiempo de respuesta de la cadena de notificación de un sistema de detección de armas basado en YOLOv8 —entre la captura del cuadro y la detección positiva (T1), la recepción en el servidor (T2), el despacho de la notificación (T3) y su entrega al dispositivo del receptor (T4)— cuando el sistema opera de forma continua en establecimientos comerciales reales, y qué eslabón concentra el retraso?

Justificación

Cada segundo que pasa entre que el arma aparece y el aviso sale cuenta para quien está en el local. El comercio de pequeña escala no tiene ese margen en un país que cerró 2023 con 8.004 homicidios y una tasa de 47,25 por cada 100.000 habitantes (Observatorio Ecuatoriano de Crimen Organizado, 2024). La cámara graba, y hasta ahí llega. Un sistema que avisa en seis segundos y otro que tarda veinte no ofrecen la misma protección. Medir es la única forma de saber cuál de los dos se tiene.

El trabajo no se agota en entrenar un detector. El autor construyó alrededor del modelo la cadena entera —captura, inferencia local, transmisión, persistencia y despacho— con marcas de tiempo en cada frontera. Así el sistema funciona como instrumento de medición y no como producto terminado. La decisión también abarata el estudio. Toda la pila es de código abierto (Python, Ultralytics YOLOv8, Django, PostgreSQL, PyQt5), las cámaras de los dos locales participantes ya estaban instaladas y validar no costó nada en infraestructura. Para un comerciante que evalúe adoptar algo así, saber dónde se pierde el tiempo define si le conviene renovar el equipo, contratar más ancho de banda o cambiar el canal de mensajería. Son tres decisiones con costos muy distintos.

La ley acota además lo que el sistema puede hacer. Toda imagen captada en un espacio comercial queda sujeta a la normativa vigente (Ley Orgánica de Protección de Datos Personales, 2021): el principio de conservación (art. 10, literal i) impide

guardar la evidencia más tiempo del que exige su finalidad; el artículo 37 obliga a proteger el acceso, y de ahí el ingreso restringido al panel web; el artículo 12 exige informar a los clientes de que hay videovigilancia. La norma no fija un plazo único de borrado para estos sistemas: cada responsable lo define y lo justifica. El sistema recogió esas obligaciones desde el diseño, no como parche posterior; el Capítulo I las desarrolla.

El proyecto cierra además el ciclo de la carrera de Ingeniería en Tecnologías de la Información de la UIDE: junta en un solo estudio de campo lo que el pénsum reparte entre visión por computadora, aprendizaje automático, sistemas distribuidos, protocolos de red, bases de datos relacionales y diseño experimental. Y deja algo utilizable para otros. Quien quiera repetir la medición con otro modelo, o con otro canal de aviso, dispone del protocolo de instrumentación tal como está.

Alcance y limitaciones

Los resultados valen para este sistema y estas condiciones, no para la detección de armas en general. El sistema operó en dos establecimientos comerciales del centro de Loja entre el 20 y el 31 de julio de 2026, con 100 ciclos completos de prueba, 50 por local. La muestra basta para caracterizar el comportamiento de la cadena. No basta para extender la conclusión a otros tipos de local ni a otras ciudades. El cliente corrió sobre un equipo de gama media: AMD Ryzen 5 4600H, 16 GB de memoria de acceso aleatorio (RAM) y unidad de procesamiento gráfico (GPU) NVIDIA GTX 1650. En un equipo más débil, o sin GPU, T1 crece.

El único canal evaluado fue el correo electrónico, enviado con la API HTTP de Brevo, así que lo concluido sobre T3 y T4 no se traslada sin más a SMS o notificaciones push. El autor cronometró T4 con las cabeceras Received, por lo que el tramo incluye la latencia de Brevo y la del servidor del destinatario. La cadena termina en el buzón: quedan fuera la apertura del mensaje y la reacción del personal.

Tampoco se recalcula el mAP@0.5 en campo, porque exigiría anotar a mano las cajas delimitadoras cuadro a cuadro. El contraste con el entrenamiento y el objetivo específico 2 se centran en precisión, recall y F1-score.

Capítulo I

Estado del Arte

1.1. Inteligencia Artificial y Visión por Computadora.

1.1.1. Fundamento de la inteligencia artificial.

¿Qué hace falta para que una máquina reconozca un arma en una pantalla? La pregunta pertenece al terreno de la inteligencia artificial (IA), que agrupa las técnicas con las que un sistema informático resuelve tareas que a una persona le exigirían percibir, razonar, aprender o decidir (International Business Machines, 2024). (Pedraza, 2023)ordena el campo en cuatro enfoques: pensar como humanos, actuar como humanos, pensar racionalmente y actuar racionalmente. El detector de este trabajo cae en el cuarto. No le interesa imitar la manera en que un guardia mira la pantalla; le basta acertar sobre un objetivo definido, la presencia de un arma en el cuadro.

Los primeros sistemas codificaban reglas escritas a mano. Ese enfoque simbólico cedió terreno al aprendizaje automático y, más tarde, al aprendizaje profundo, donde el algoritmo saca los patrones de los datos sin que nadie los enuncie. De ahí salieron las redes neuronales convolucionales (CNN), que aprenden características visuales por capas: bordes y texturas en las primeras, formas completas en las profundas (Google Cloud, 2024). YOLOv8, el modelo de este trabajo, hereda esa arquitectura.

Nada de esto avanzó en línea recta. Hubo «inviernos»: períodos de expectativas altas que terminaron en recortes de financiamiento y años de estancamiento (Librería CATEDU, 2023).). La Figura 2 recorre esa línea de tiempo desde la conferencia de Dartmouth, en 1956, hasta el auge actual. Dos cosas coincidieron para producirlo. Aparecieron volúmenes de datos etiquetados que antes no existían, y las GPU abarataron el entrenamiento de redes profundas de semanas a horas.

Figura 2:

Evolución del desarrollo de la IA, desde los enfoques simbólicos hacia el aprendizaje automático y profundo.



Nota. Adaptado de (Librería CATEDU, 2023).

1.1.2. ¿Qué es la visión por computadora?

La visión por computadora vive entre dos campos: la IA y el procesamiento digital de imágenes. Busca que una máquina interprete información visual con una fiabilidad comparable a la del ojo humano; en tareas acotadas, incluso mayor (Mirmehdi, 2020). (Fraile, 2024) enumera sus aplicaciones habituales, de la detección de objetos y la clasificación al reconocimiento facial, el seguimiento de movimiento y la reconstrucción tridimensional de escenas. Todas comparten un mismo problema de fondo: convertir una matriz de píxeles en una descripción de lo que hay delante de la cámara. Este trabajo se queda con la primera aplicación de la lista.

El proceso encadena cinco etapas. Una cámara o un sensor adquiere la imagen; el preprocesamiento reduce el ruido y la normaliza; la segmentación recorta las regiones

de interés; la extracción de características las describe, y un modelo de aprendizaje interpreta el resultado.

Los detectores de una sola pasada, con YOLO al frente, funden las tres etapas centrales en una sola inferencia de la red. El salto no fue solo algorítmico. Se volvió viable cuando coincidieron el hardware especializado, GPU y TPU, con datasets etiquetados del tamaño de ImageNet y COCO.

El procesamiento digital clásico exigía programar reglas explícitas para cada tipo de objeto. Un sistema de aprendizaje profundo se salta ese paso: aprende por sí mismo las representaciones visuales que necesita para detectar y clasificar.

1.1.3. Revolución de la visión por computadora a través de la IA.

El aprendizaje profundo reordenó la disciplina de arriba abajo. Antes de las CNN, quien construía un sistema de visión diseñaba a mano sus descriptores, SIFT y HOG entre los más usados. Ese diseño manual era el punto débil: un cambio de iluminación, un ángulo nuevo o una oclusión parcial bastaban para degradar el resultado. Las redes entrenadas de extremo a extremo eliminaron el paso. Aprenden sus propias representaciones jerárquicas desde los píxeles y toleran variaciones que los descriptores manuales nunca soportaron (Miao, 2021). Se paga con opacidad. Nadie puede leer qué aprendió la red.

(Brodowicz, 2024) atribuye la revolución a tres condiciones que llegaron juntas: grandes volúmenes de datos debidamente etiquetados, arquitecturas de redes profundas capaces de aprovecharlos y la potencia de cálculo para entrenarlas, que apareció con las GPU.

Con esas tres piezas en su lugar maduraron la detección de objetos en tiempo real, que es el terreno de YOLO, junto con la segmentación semántica y la conducción autónoma.

Con esas tres piezas en su lugar maduraron la detección de objetos en tiempo real, que es el terreno de YOLO, junto con la segmentación semántica y la conducción autónoma.

Tabla 1:

Comparación entre visión por computadora tradicional y visión por computadora basada en IA

Característica	Enfoque Tradicional	Enfoque basado en IA (Deep Learning)
Extracción de características	Manual (SIFT, HOG, Haar)	Automática (aprendida por la red)
Robustez ante variaciones	Baja (sensibles a ángulos/iluminación)	Alta (aprenden invariancias desde los datos)
Necesidad de expertise técnica	Alta (diseño de descriptores específicos)	Media (requiere arquitectura y entrenamiento)
Precisión	Moderada	Alta
Tiempo de entrenamiento	Bajo	Alto
Escalabilidad	Limitada	Muy alta

Nota. Adaptado de (Brodowicz, 2024).

Las dos columnas se separan en un punto: quien diseña las características. En el enfoque tradicional las define un experto, descriptor por descriptor, y ese diseño hay que rehacerlo cuando cambian el ángulo de cámara o la iluminación. La red profunda las aprende de los datos y absorbe esa variación durante el entramiento, a cambio de un costo de cómputo más alto.

1.2. Técnicas de IA en Visión por Computadora.

1.2.1. Redes Neuronales Convolucionales (CNN): estructura y aprendizaje.

Las CNN son la arquitectura sobre la que descansa la visión por computadora actual. Su diseño parte de un supuesto sobre los datos: las imágenes tienen estructura

de cuadrícula y los píxeles vecinos guardan relación entre sí. (De la Cruz, 2024) rastrea esa idea hasta la organización jerárquica de la corteza visual humana, donde el procesamiento avanza de rasgos simples a representaciones complejas.

Tres tipos de capa se reparten el trabajo:

- Convolucionales: Deslizan filtros -kernels- sobre la imagen y producen mapas de características que responde a bordes, texturas y patrones locales. Los pesos del filtro no se diseñan a mano: se ajustan durante el entramiento.
- Capas de pooling: Reducen la dimensionalidad mediante operaciones como max pooling o average pooling, manteniendo la información esencial.
- Totalmente conectadas (fully connected). Cierran la cadena. Toman las características ya extraídas y las convierten en la decisión de clase.

Durante el entrenamiento, las CNN ajustan automáticamente los pesos de sus filtros a través del algoritmo de retro propagación (backpropagation), minimizando la diferencia entre la predicción y la etiqueta real mediante técnicas de optimización como el descenso de gradiente estocástico. Este proceso permite que las redes aprendan a identificar patrones visuales relevantes directamente desde los datos, sin necesidad de diseñar manualmente los descriptores (Forruque, 2023).

La Tabla 2 resume las funciones y características de las capas principales que componen esta arquitectura:

Tabla 2:

Componentes fundamentales de una red neuronal convolucional

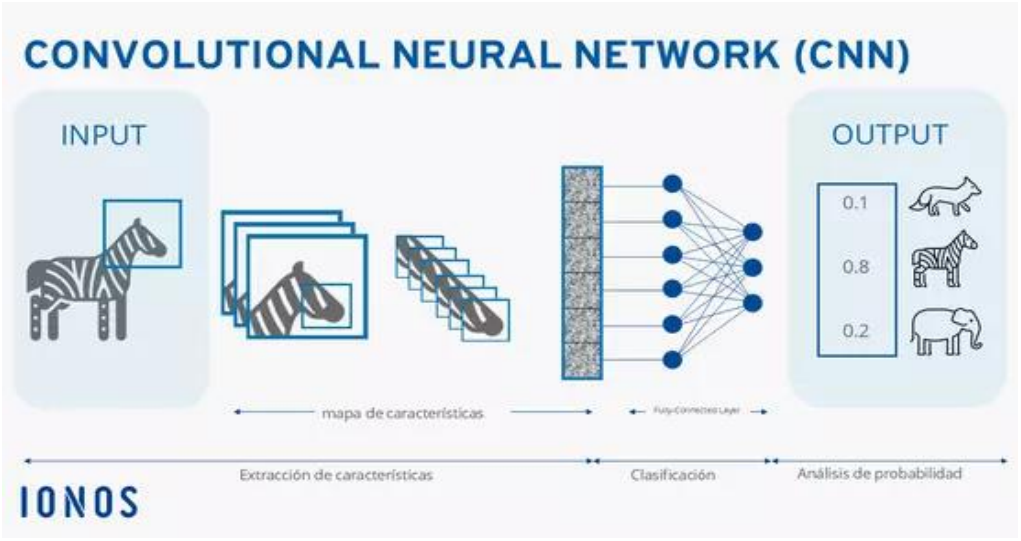
Tipo de capa	Función principal	Resultado esperado
Convolucional	Extraer características locales	Mapas de activación (bordes, texturas, patrones)
Pooling (submuestreo)	Reducir la dimensionalidad y preservar información	Invarianza espacial, reducción de complejidad
Fully connected	Tomar decisiones finales	Clasificación o predicción
Normalización y dropout	Estabilizar y regularizar el aprendizaje	Mejor generalización, reducción de sobreajuste

Nota. Adaptado de (IONOS Digital Guide, 2024).

La Figura 3 ilustra el flujo de procesamiento de una CNN típica, desde la imagen de entrada hasta la predicción final. En ella se puede observar cómo se extraen mapas de características a través de capas convolucionales, los cuales son posteriormente procesados por capas densas (fully connected) para emitir una clasificación probabilística sobre las posibles categorías del objeto analizado.

Figura 3:

Arquitectura básica de una red neuronal convolucional (CNN): entrada, extracción de características, clasificación y salida probabilística.



Nota. Adaptado de (IONOS Digital Guide, 2024).

Este diseño jerárquico permite que las CNN aprendan representaciones cada vez más abstractas a medida que se avanza hacia capas más profundas, lo que las hace especialmente eficaces en tareas como detección de objetos, reconocimiento facial y segmentación de imágenes.

1.2.2. Modelos R-CNN, Fast R-CNN, YOLO: evolución y principios

La evolución de los modelos de detección de objetos ha transitado desde enfoques lentos y complejos hacia arquitecturas optimizadas para aplicaciones en tiempo real (Jiménez, 2025). A continuación, se describen los principales hitos en esta evolución.

- **R-CNN (Regions with CNN features):** Este modelo fue pionero al combinar propuestas de regiones con redes neuronales convolucionales. Primero generaba unas 2000 regiones candidatas usando Selective Search y luego aplicaba una CNN a cada región, lo cual ofrecía buena precisión, pero era computacionalmente costoso.
- **Fast R-CNN:** Propone procesar toda la imagen con una CNN y luego usar una capa de RoI pooling para extraer características de cada región propuesta. Esto redujo significativamente el tiempo de inferencia y mejoró la eficiencia.
- **Faster R-CNN:** Incorpora una red de propuestas de regiones (RPN), eliminando la dependencia de Selective Search y acelerando aún más la detección. Es sumamente preciso, pero no alcanza velocidad en tiempo real aún.
- **YOLO (You Only Look Once):** Rompe con el esquema de dos etapas. En lugar de proponer regiones y clasificarlas después, trata la detección como un problema de regresión único: la red divide la imagen en una cuadrícula y, en una sola pasada, predice las bounding boxes junto con la clase de cada objeto. Esa pasada única es lo que abre la puerta al tiempo real.

La Tabla 3 reúne cifras de velocidad, precisión y mecanismo de generación de regiones de las cuatro arquitecturas.

Tabla 3:

Comparación entre modelos R-CNN, Fast R-CNN, Faster R-CNN y YOLO

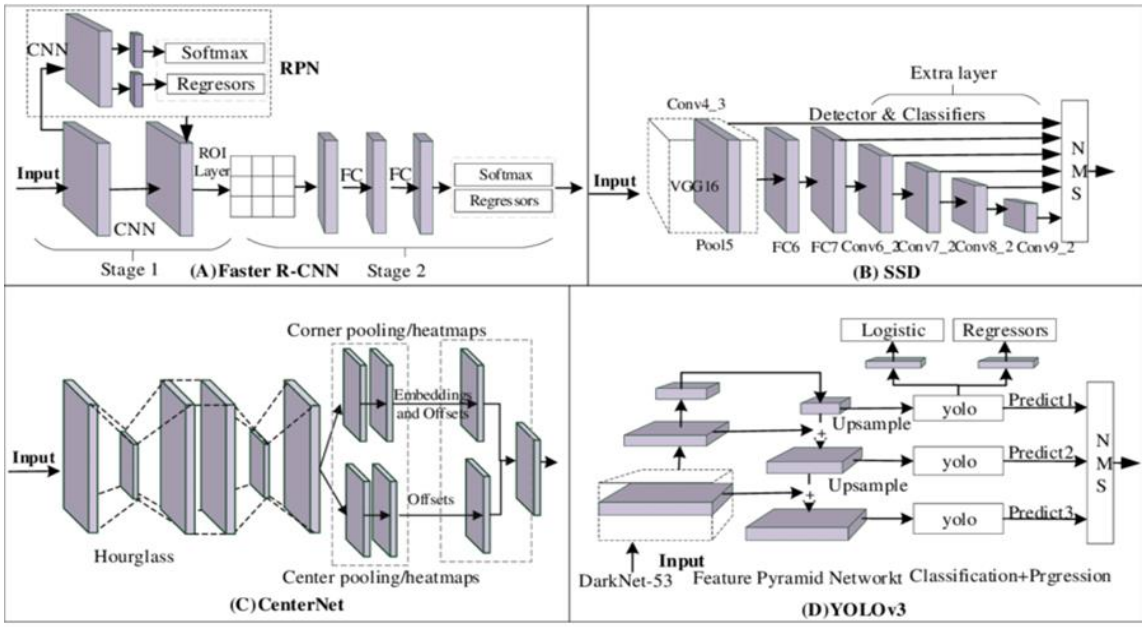
Modelo	Velocidad (FPS)	Precisión	Generación de regiones	Adecuado para tiempo real
R-CNN	0.5	Alta	Selective Search externa	No
Fast R-CNN	1	Alta	Selective Search externa	No
Faster R-CNN	5–7	Muy alta	RPN integrada	Parcialmente
YOLOv1-v8	30–300+	Alta–Muy alta	Detección unificada	Sí

Nota. Adaptado de (Jiménez, 2025). FPS = cuadros por segundo.

A este trabajo le interesa un contraste puntual: Faster R-CNN frente a YOLO. Con 5 a 7 cuadros por segundo, Faster R-CNN alcanza para analizar video en diferido. Un local comercial pide otra cosa. La cámara entrega 30 cuadros por segundo y la alerta tiene que salir mientras el arma sigue en escena, lo que obliga a inferir en menos de 33 ms por cuadro. YOLO cumple esa cota, y por eso el sistema evaluado aquí usa YOLOv8.

Figura 4:

Arquitecturas de detección de objetos: Faster R-CNN, SSD (Single Shot Detector), CenterNet y YOLO.



Nota. Adaptado de (Hong, 2021).

La figura deja ver una dirección: cada generación recortó el costo de cómputo sin ceder precisión. De ese equilibrio depende que la videovigilancia en tiempo real sea viable, y con ese mismo criterio este trabajo eligió detector. Ganar precisión perdiendo velocidad no sirve de nada si la alerta llega tarde.

1.2.3. Procesamiento de imágenes y video para detección de objetos.

Ningún detector supera la calidad de la imagen que recibe. Por eso cada cuadro pasa, antes de la inferencia, por operaciones que compensan lo que la cámara no controla —iluminación irregular, ruido del sensor, resolución corta— y acotan las regiones donde conviene buscar (Reyes, 2020).

Ese preprocesamiento corrige la iluminación, reduce el ruido y escala la imagen al tamaño de entrada de la red. La segmentación parte después la escena en regiones con significado propio. Los operadores clásicos siguen vigentes para trazar contornos: Sobel y Canny lo resuelven con un costo de cómputo bajo.

En el otro extremo aparecen las mejoras por aprendizaje profundo: superresolución con SRGAN, normalización de color, realce de contraste. Recuperan detalle. También cobran, y cobran en milisegundos que salen del presupuesto de latencia.

El flujo de video agrega una exigencia que la imagen fija no tiene: no se detiene. Cada segundo llegan decenas de cuadros que compiten por memoria, ancho de banda y tiempo de inferencia. El sistema decide entonces qué procesa y qué descarta, sin dejar pasar el evento que importa. Con un preprocesamiento eficiente, un detector como YOLO mantiene tasas altas de cuadros por segundo sin sacrificar exactitud (Amazon Web Services, 2024).

La Tabla 4 reúne las técnicas más usadas, con su objetivo y su papel en detección.

Tabla 4:

Técnicas comunes de procesamiento aplicadas a imágenes y video para visión por computadora

Técnica	Objetivo principal	Aplicación en detección
Normalización	Estandarizar datos de entrada	Facilita aprendizaje eficiente
Detección de bordes	Resaltar contornos de objetos	Segmentación inicial
Filtro Gaussiano	Eliminar ruido	Suavizado previo a detección
Superresolución	Aumentar calidad de imágenes	Mejorar reconocimiento de armas
Submuestreo de cuadros	Reducir procesamiento sin perder contexto	Flujo de video en tiempo real

Nota. Adaptado de (Amazon Web Services, 2024).

De ese diseño unificado sale la ventaja de velocidad frente a los métodos de dos etapas. En vigilancia automatizada eso decide algo concreto: o la alerta llega durante el evento, o llega después.

1.3. Modelos YOLO y su Aplicación en Tiempo Real.

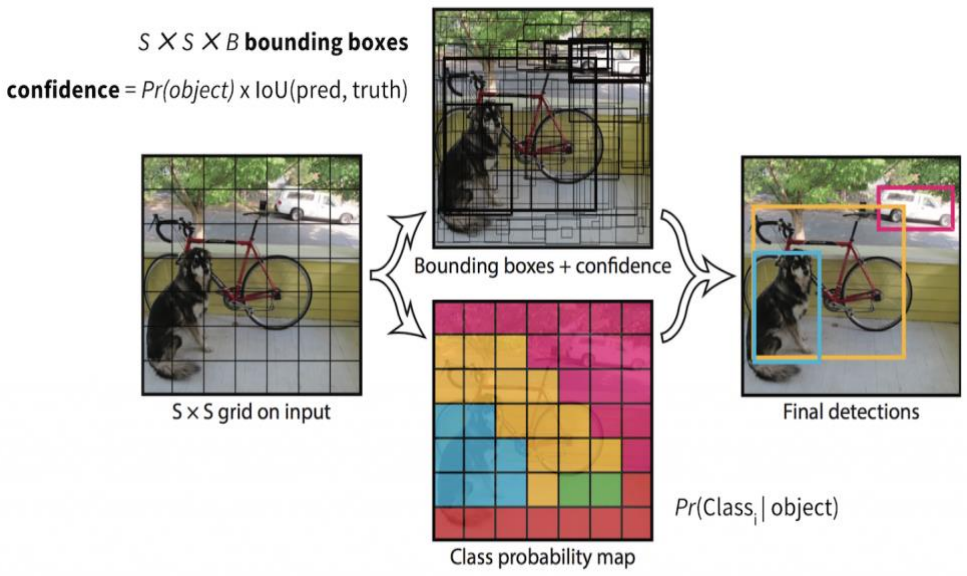
1.3.1. Principios y arquitectura de YOLO.

El modelo YOLO (You Only Look Once) representa un cambio gigantesco en la detección de objetos mediante visión por computadora. A diferencia de enfoques como R-CNN o SSD, que dividen el proceso en múltiples etapas (propuesta de regiones, clasificación, ajuste de bounding boxes), YOLO plantea la detección como un solo problema de regresión, resolviéndolo en una única pasada por la red neuronal.

YOLO parte la imagen de entrada en una cuadrícula S×S y predice, por cada celda, B cajas delimitadoras (bounding boxes) con la probabilidad de cada clase (Figura 5). Una función sigmoide procesa esas predicciones. Después, la supresión de no máximos (Non-Maximum Suppression, NMS) elimina los duplicados y solo sobreviven las detecciones más confiables.

Figura 5:

Detección de objetos con YOLO



Nota. Adaptado de (Virtudes, 2023).

Esa pasada única es la fuente de la velocidad. YOLO infiere varias veces más rápido que las arquitecturas de dos etapas, y para la vigilancia automatizada y el reconocimiento de amenazas en entornos urbanos esa diferencia separa reaccionar de solo registrar.

1.3.2. Avances en las versiones YOLOv5 y YOLOv8

YOLOv8 cambió el backbone por uno propio de Ultralytics y dejó las anclas: su cabeza anchor-free predice el centro de cada objeto sin cajas de referencia. También sumó segmentación y estimación de pose a las tareas soportadas. La Tabla 5 compara ambas versiones.

Tabla 5:

Comparación técnica entre YOLOv5 y YOLOv8

Característica	YOLOv5	YOLOv8
Arquitectura backbone	CSPDarknet53	Backbone Ultralytics propio
Tipo de detección	Basado en anclas (	*Anchor-free*
Framework principal	PyTorch	PyTorch
Tareas soportadas	Detección	Detección, segmentación, poses
Formato de salida	Bounding boxes	Bounding boxes y máscaras
Curva de aprendizaje	Rápida	Optimizada
Precisión (mAP@0.5)	Alta	Muy alta
Velocidad de inferencia	30–60 FPS	50–90 FPS (según configuración)

Nota. Adaptado de (Ultralytics, 2024).

YOLOv5 incorporó el backbone CSPDarknet53, el ajuste automático de anclas y aumentos de datos como Mosaic y MixUp. Escrita de forma nativa en PyTorch, bajó la barrera de entrada: entrenar sobre un dataset propio dejó de pedir dominio profundo del framework. Ahí se explica buena parte de su adopción en proyectos aplicados.

Este trabajo empleó YOLOv8 para el sistema desplegado en Loja. La versión corre entre 50 y 90 cuadros por segundo según configuración y, tras el fine-tuning sobre el dataset propio de 4.000 imágenes, alcanzó 90,3 % de precisión, 89,4 % de recall y 91,2 % de mAP@0.5 en entrenamiento.

1.3.3. Ventajas de YOLO para detección en tiempo real

La ventaja de fondo ya quedó dicha: una sola evaluación de red por imagen, sin etapas intermedias que sumen latencia. De ahí se desprenden las demás.

Entre las principales ventajas de YOLO se destacan:

- La velocidad de inferencia va de 30 a 300 FPS según versión y hardware.
- Corre en hardware embebido. Existen variantes optimizadas para dispositivos de borde (Edge computing) como Jetson Nano, Raspberry Pi o Coral TPU, lo que abre el despliegue sin servidor dedicado.
- Detecta múltiples clases de objeto en la misma pasada, sin reentrenar por escena.
- El consumo de recursos en ejecución es bajo, condición que en un local comercial con un equipo de gama media marca la diferencia entre procesar el flujo completo de la cámara o descartar cuadros.

Sobre esa base se han montado sistemas de alerta por armas, monitoreo de multitudes y control vehicular. Conviene precisar el incidente sigue siendo humana. Lo que la IA hace es recortar el tiempo entre la aparición de la amenaza y el aviso a la persona exactamente la cadena que esta investigación mide.

1.4. Aplicaciones de IA en Sistemas de Seguridad y Vigilancia

1.4.1. Casos de éxito en videovigilancia basada en IA

Los despliegues reales de videovigilancia con IA ya dejaron resultados medibles en tres continentes.

El caso de mayor escala es Hangzhou, en China: más de un millón de cámaras interconectadas con algoritmos de reconocimiento facial y detección de anomalías, que redujeron el tiempo promedio de resolución de casos criminales y terminaron usándose también para gestión de tráfico y control de multitudes (Arana, 2019).

Barcelona optó por un alcance más acotado. Su sistema analiza video en tiempo real para identificar actos vandálicos y violencia en espacios públicos, y (Hernández, 2025) documenta que el filtro de falsos positivos fue lo que hizo viable la operación: sin él, cada falsa alarma consumía recursos policiales.

En América Latina la adopción va por pilotos. Ciudad de México y Buenos Aires probaron YOLOv5 junto con DeepStream de NVIDIA para reconocimiento de armas, seguimiento de sospechosos y análisis de flujo vehicular en zonas de alta densidad poblacional (Nvidia, 2025). Ninguno de esos pilotos, hasta donde reporta la fuente, midió la cadena de notificación posterior a la detección — el vacío que este trabajo cubre a escala de comercio local.

1.4.2. Algoritmos y plataformas para monitoreo inteligente

El ecosistema de videovigilancia inteligente actual combina distintos componentes tecnológicos, desde cámaras IP conectadas a redes seguras, hasta algoritmos de aprendizaje profundo desplegados en servidores locales o en la nube. Estos algoritmos permiten no solo identificar objetos (personas, vehículos, armas), sino también analizar comportamientos, realizar conteos automáticos y generar alertas en tiempo real (Unapucha, 2025).

Entre las plataformas más utilizadas para este tipo de soluciones se encuentran:

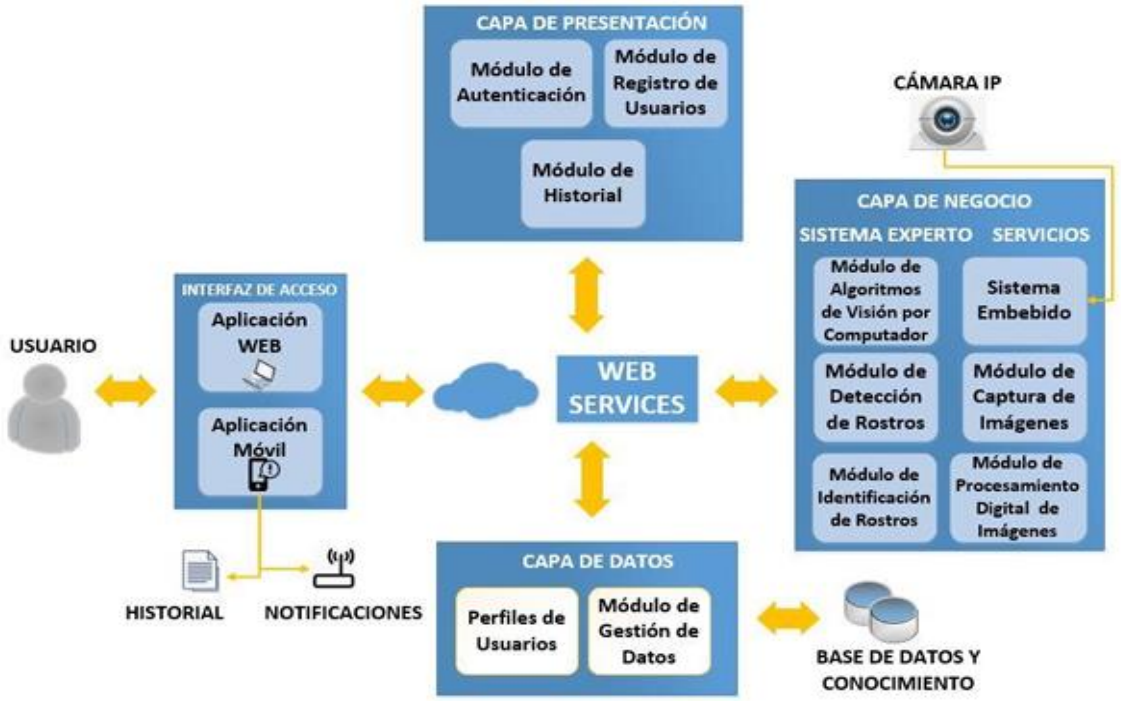
- **NVIDIA DeepStream SDK:** permite inferencias en tiempo real sobre múltiples flujos de video en hardware optimizado (Jetson, Tesla, etc.).
- **Ultralytics YOLOv5/yolov8:** ampliamente usado por su velocidad, precisión y facilidad de integración.

A nivel de software, lenguajes como Python junto con frameworks como PyTorch, TensorFlow y bibliotecas como OpenCV constituyen la base de los sistemas desarrollados a medida.

En la Figura 6 se ilustra la arquitectura general de un sistema de videovigilancia inteligente basado en IA, en la que se observa cómo los distintos elementos desde las cámaras IP y dispositivos de borde, hasta los servidores y paneles de control trabajan de forma coordinada para procesar el flujo de video en tiempo real y emitir alertas precisas ante eventos críticos.

Figura 6:

Arquitectura general de un sistema de videovigilancia inteligente basado en IA



Nota. Adaptado de (Gutiérrez, 2021).

1.4.3. Desafíos actuales: privacidad y robustez

A pesar de sus múltiples beneficios, la implementación de IA en videovigilancia plantea desafíos éticos, técnicos y sociales. El primer reto es la privacidad. Que la cámara esté en un espacio público no vuelve públicos los datos que capta: rostros, movimientos

y objetos personales siguen exigiendo reglas claras que impidan el abuso y la vigilancia masiva sin control (Pérez, 2021).

Queda el problema de la solidez del detector. La iluminación y el clima degradan su rendimiento, y existen ataques adversariales pensados para engañarlo: un patrón impreso en la ropa puede suprimir o distorsionar la detección.

- Contra esos frentes se trabajan enfoques complementarios:
- Anonimización en tiempo real: desenfoque o pixelado de rostros no relevantes.
- Auditoría algorítmica: registro de decisiones del modelo y trazabilidad.
- Entrenamiento con datos variados y adversariales: para aumentar la resiliencia del sistema ante falsos positivos o manipulación.

Ningún desafío de esta lista se resuelve solo con código. Hace falta cruzar el desarrollo técnico con la legislación, la ética computacional y la participación ciudadana para que la vigilancia inteligente cumpla su propósito sin pisar derechos fundamentales.

1.5. Detección de Armas Mediante Inteligencia Artificial

1.5.1. Complejidades técnicas: variedad, oclusión, entorno

Detectar armas en video es de lo más difícil que se le puede pedir a la visión por computadora. (Gutiérrez Velázquez, 2023) reúne las razones: a diferencia de una persona o un vehículo, el arma cambia mucho de forma entre ejemplares, ocupa pocos píxeles en la escena y suele aparecer parcialmente oculta o en la mano de alguien que se mueve.

Entre las principales dificultades técnicas destacan:

- **Variedad de modelos y formas:** pistolas, revólveres, fusiles, cuchillos, armas hechizas, etc., cada una con características visuales distintas.
- **Oclusión parcial o total:** cubiertas por ropa, bolsas o manos, dificultando su detección directa.

- **Condiciones ambientales:** iluminación deficiente, baja resolución, movimiento de cámara o ruido visual en escenarios reales.

Estos factores exigen sistemas de detección entrenados con datasets diversos, técnicas de aprendizaje robustas y capacidades de adaptación a condiciones variables. La Tabla 6 resume las principales complejidades enfrentadas por los modelos de IA en la detección de armas en ambientes urbanos y públicos.

Tabla 6:

Principales desafíos técnicos en la detección de armas mediante IA

Complejidad técnica	Descripción	Efecto en la detección
Variabilidad morfológica	Armas de diferentes tamaños, colores, texturas y formas	Aumenta tasa de falsos negativos
Oclusión	Armas cubiertas parcial o totalmente por el cuerpo u objetos	Reducción de precisión
Escala y resolución	Armas pequeñas en fondo de imagen o baja calidad de video	Disminuye confianza del modelo
Escenario dinámico	Movimiento de personas, cámaras o multitudes	Reto para el seguimiento y enfoque

Nota. Adaptado de (Villarreal, 2023).

1.5.2. Datasets específicos y técnicas de aumento de datos

El entrenamiento de modelos de detección de armas requiere datasets especializados que contengan imágenes etiquetadas con armas visibles en diversas posiciones y contextos. A diferencia de datasets genéricos como COCO o ImageNet, los conjuntos de datos de armas suelen ser limitados, lo que obliga a utilizar estrategias de aumento de datos (data augmentation) para mejorar la generalización del modelo.

Algunos datasets relevantes para esta tarea incluyen:

- Gun Detection Dataset (Open Images - armas): contiene imágenes etiquetadas con diferentes tipos de armas de fuego.
- Flir Thermal Dataset: útil en detección nocturna o con visión térmica.
- Synthetic Gun Datasets: generados artificialmente con renders 3D para aumentar la variabilidad visual.
- Custom datasets: elaborados por instituciones para contextos locales (vigilancia urbana, escuelas, etc.).

Para compensar la escasez de datos reales, se aplican técnicas como:

- Rotación y escalado: simula diferentes ángulos de visión.
- Ajuste de brillo/contraste: adapta el modelo a condiciones lumínicas variables.
- Inserción sintética de armas en fondos reales: genera imágenes compuestas más realistas.

En la Tabla 7 se resumen algunos datasets específicos y sus características.

Tabla 7:

Datasets relevantes para detección de armas con IA

Nombre del dataset	Tipo de armas	Contexto	Observaciones
Open Images (subconjunto)	Pistolas, cuchillos	Diverso	Etiquetado en COCO format
FLIR Thermal	Desconocido	Térmico/seguridad	Imágenes térmicas nocturnas
Gun Dataset (Roboflow)	Pistolas	Escuelas, oficinas	Muy útil para transfer learning
Synthetic Weapon Dataset	Variado (3D)	Generado	Ideal para data augmentation inicial

Nota. Adaptado de (Villarreal, 2023).

1.5.3. Transfer learning y fine-tuning en modelos de armas

Dada la escasez de grandes datasets específicos, una estrategia eficaz es aplicar transfer learning, es decir, utilizar un modelo previamente entrenado en un dataset genérico (como COCO o PASCAL VOC) y ajustarlo (fine-tune) a un conjunto más pequeño de imágenes específicas de armas.

Esta técnica aprovecha las características generales ya aprendidas por la red (bordes, formas, texturas) y simplemente reajusta las capas finales para reconocer objetos particulares como armas. Modelos como YOLOv5 y YOLOv8 soportan fácilmente este enfoque, permitiendo:

- Reducir tiempo de entrenamiento.
- Disminuir requerimientos computacionales.
- Mejorar la precisión con menos datos.

Para ello se suelen congelar las capas iniciales del modelo y reentrenar solo las capas de salida con el nuevo dataset etiquetado. Esta técnica ha demostrado ser especialmente útil para contextos locales donde los modelos deben adaptarse a características visuales del entorno (uniformes, tipos de arma, iluminación regional, etc.).

1.5.4. Métricas de evaluación: precisión, recall, mAP, F1-score

Para evaluar el desempeño de un modelo de detección de armas, se emplean métricas estándar en visión por computadora que reflejan la capacidad del sistema para detectar objetos correctamente y con fiabilidad (Jayaswal, 2020). Las principales métricas utilizadas incluyen:

- **Precisión (Precision):** proporción de detecciones verdaderas sobre el total de predicciones positivas.
- **Recall:** proporción de detecciones verdaderas sobre el total de armas reales presentes.

- **F1-score:** media armónica entre precisión y recall, útil para evaluar el balance general.
- **mAP:** promedio de la precisión en diferentes umbrales de intersección sobre unión (Intersection over Union, IoU), especialmente mAP@0.5 y mAP@0.5:0.95.

La Tabla 8 resume estas métricas clave y su interpretación práctica en el contexto de vigilancia inteligente. En las fórmulas, TP son los verdaderos positivos, FP los falsos positivos y FN los falsos negativos.

Tabla 8:

Métricas de evaluación para modelos de detección de armas.

Métrica	Fórmula	Interpretación
Precisión	TP / (TP + FP)	Cuántas detecciones fueron correctas
Recall	TP / (TP + FN)	Cuántos objetos reales fueron detectados
F1-score	2 × (Precisión × Recall) / (P + R)	Balance entre precisión y cobertura
mAP@0.5	Media de precisión con IoU ≥ 0.5	Precisión general sin ser demasiado exigente
mAP@0.5:0.95	Promedio de mAP entre IoU 0.5 y 0.95	Evaluación más rigurosa y completa

Nota. Adaptado de (Jayaswal, 2020).

1.5.5. Validación en escenarios reales y selección de umbrales

Las métricas de laboratorio no bastan. El modelo entrenado tiene que probarse bajo las condiciones que va a encontrar en operación: cámaras que vibran, iluminación que cambia con la hora, multitudes, y objetos que se parecen a un arma —un taladro, una pistola de juguete— sin serlo.

En esta etapa se fija el umbral de confianza que separa una detección válida de una descartada, casi siempre entre 0,6 y 0,8. Elegir el umbral es negociar. Si baja, sube

el recall y suben también los falsos positivos; si sube, la precisión mejora, pero un arma real detectada con 0,55 de confianza pasaría sin disparar la alerta. Tres herramientas ubican ese punto de operación: las curvas Precision-Recall, que recorren el rendimiento en todos los umbrales; el análisis de sensibilidad sobre los parámetros establecidos; y las pruebas controladas en campo con actores y réplicas inertes. Realizar el despliegue antes de esa calibración no tiene sentido.

1.5.6. El tiempo de respuesta en los sistemas de alerta: lo que se mide y lo que no

¿Qué mide la literatura cuando dice que un sistema es rápido? Mide el modelo: cuántos milisegundos por cuadro, cuántos cuadros por segundo. Así lo recoge la Tabla 3 de este capítulo, con 0,5 FPS en R-CNN, entre 5 y 7 en Faster R-CNN y más de 30 en la familia YOLO (Jiménez, 2025). El mismo criterio usa Ultralytics (2024) para comparar YOLOv5 con YOLOv8, al que sitúa entre 50 y 90 FPS según configuración.

Todas esas cifras describen un solo eslabón. El detector que infiere en 30 milisegundos no avisa a nadie en 30 milisegundos: todavía falta guardar la evidencia, enviarla al servidor, procesarla en el backend, despachar la notificación y entregarla en el dispositivo del destinatario. Ninguno de esos tramos obedece al modelo. Obedecen al ancho de banda del local, a la carga del servidor y a la cola del proveedor de correo.

Ese trayecto no aparece medido en ninguno de los trabajos revisados. (Unapucha, 2025) presenta el antecedente más próximo a esta investigación: un sistema de circuito cerrado de televisión (CCTV) con alerta temprana y reconocimiento de armas de fuego que reporta el desempeño del clasificador y nada más. Gutiérrez (2021) sí documenta la arquitectura completa hasta la alerta sobre dispositivos embebidos, pero no cronometra lo que pasa después de la inferencia. (Hernández, 2025) revisa sistemas de vigilancia con IA para prevenir el delito y discute el filtrado de falsos positivos como condición operativa; la latencia del aviso no entra en su análisis. Y en los pilotos latinoamericanos que documenta (Nvidia, 2025) el foco está en reconocer y seguir, no en avisar.

Hay una razón para ese vacío. Cronometrar la inferencia cuesta poco: se rodea una llamada a función con dos marcas de tiempo, sobre un equipo controlado y un dataset fijo. Cronometrar la cadena entera cuesta mucho más, porque obliga a instrumentar tres máquinas, sincronizar sus relojes y sostener el sistema operando en un sitio real durante días. El costo explica el hueco en la literatura; no lo justifica.

La misma zona ciega aparece en quienes estudian los límites operativos de la videovigilancia automatizada. El rendimiento de laboratorio y el de despliegue divergen por factores de entorno (Pérez, 2021). En el extremo contrario, el sistema de Hangzhou acortó el tiempo de resolución de casos criminales (Arana, 2019), un indicador que se cuenta en horas de expediente y no en segundos de aviso. En medio quedan los segundos que nadie ha caracterizado. Son los segundos en que el personal de un local decide si alcanza a reaccionar.

La Tabla 9 muestra qué reporta cada trabajo revisado y hasta dónde llega su medición.

Tabla 9

Alcance temporal de las mediciones reportadas en los trabajos revisados

Trabajo	Qué reporta	Tramo cubierto
(Jiménez, 2025)	FPS por arquitectura	Inferencia
(Ultralytics, 2024)	FPS y mAP por versión	Inferencia
(Unapucha, 2025)	Métricas del clasificador	Inferencia
(Gutiérrez, 2021)	Arquitectura del sistema	Sin medición temporal
(Hernández, 2025)	Tasa de falsos positivos	Sin medición temporal
(Nvidia, 2025)	Reconocimiento y seguimiento	Inferencia
(Arana, 2019)	Tiempo de resolución de casos	Posterior a la alerta

Nota. Elaboración propia a partir de las fuentes revisadas.

Este trabajo se instala en ese rango. Ninguna de las fuentes de la Tabla 9 desagrega los cuatro tramos que aquí se miden, T1 a T4.

1.6. Consideraciones Éticas y Legales en la Implementación

Apuntar cámaras a personas no es solo un asunto técnico. Toca derechos fundamentales: la imagen, la privacidad, la presunción de inocencia. La legitimidad del sistema depende de que los respete, de que transparente sus procesos y de que opere dentro de la normativa vigente. En Ecuador esa normativa existe y es reciente.

1.6.1. Protección de datos y privacidad en vigilancia digital

Lo más delicado es el tratamiento de datos personales. Una grabación de seguridad parece material genérico. No lo es: identifica rostros, placas y rutinas de comportamiento, y eso la convierte en dato personal según la normativa ecuatoriana (Ley Orgánica de Protección de Datos Personales, 2021). La ley reconoce a los ciudadanos el derecho a acceder a sus datos, rectificarlos y oponerse a su uso indebido, y prohíbe la videovigilancia masiva sin justificación legal o judicial.

De ahí salen los principios que un sistema de videovigilancia debe cumplir. La captación necesita un objetivo legítimo, como prevenir delitos (finalidad específica). Solo se graba lo necesario y por un tiempo limitado (minimización de datos). La información viaja cifrada y con acceso restringido, y las personas tienen derecho a saber que las están grabando, lo que en la práctica es un letrero visible en el local.

1.6.2. Prevención de sesgos algorítmicos en seguridad pública

El sesgo algorítmico es el segundo frente. Un modelo puede fallar más sobre ciertos grupos de la población, y las causas del desbalance son conocidas:

- Datos de entrenamiento no representativos: un dataset dominado por un grupo étnico o geográfico produce un modelo que generaliza mal sobre los demás.
- Diseño del modelo: hay arquitecturas que se inclinan a detecciones erróneas según la ropa, los gestos, los objetos o el entorno.
- Retroalimentación errónea: cuando el sistema aprende de decisiones humanas ya sesgadas, hereda y perpetúa esos errores.

Estos sesgos pueden traducirse en falsas alarmas frecuentes en determinados sectores, persecución injustificada o afectación de derechos, especialmente en contextos de alta vigilancia. Para mitigar este riesgo, se recomienda:

- Aplicar técnicas de balance de datos durante el entrenamiento.
- Usar auditorías algorítmicas periódicas.
- Promover transparencia en las decisiones automatizadas.
- Incorporar principios de IA ética, como los propuestos por UNESCO y la Comisión Europea.

1.6.3. Normativas y marco regulatorio nacional e internacional

La regulación de sistemas de vigilancia con IA está en construcción en la mayoría de países. A nivel internacional, existen lineamientos importantes como:

- **UNESCO – Recomendación sobre la Ética de la IA (2021):** establece principios de transparencia, justicia, seguridad y respeto por los derechos humanos.
- **IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems**: promueve estándares éticos para crear sistemas inteligentes seguros y justos.

La Tabla 10 resume los marcos normativos clave que deben considerarse en el diseño e implementación de sistemas de detección de armas en el espacio público.

Tabla 10:

Principales marcos normativos aplicables a la vigilancia con IA

Marco normativo	Alcance	Relevancia para IA y vigilancia
Ley Orgánica de Protección de Datos (ECU)	Nacional – Ecuador	Regula tratamiento de imágenes, consentimiento y derechos ciudadanos
RGPD – Unión Europea	Internacional	Establece principios globales de protección de datos y transparencia
UNESCO – Ética de la IA (2021)	Internacional	Guía ética para el diseño responsable de IA en vigilancia pública
IEEE AI Standards	Internacional (tecnológico)	Estándares técnicos para evitar daños y sesgos algorítmicos

Nota. Adaptado de (Ley Orgánica de Protección de Datos Personales, 2021).

En el caso de Ecuador, la Constitución (art. 66, num. 19) reconoce el derecho a la protección de datos personales, y la Ley Orgánica de Protección de Datos Personales (LOPDP) entró en vigencia en 2021. Esta ley obliga a los responsables de tratamiento (como instituciones que manejan sistemas de vigilancia) a implementar evaluaciones de impacto, registrar sus bases de datos, y respetar los principios de legalidad, lealtad, proporcionalidad y responsabilidad proactiva.

Si bien los sistemas basados en IA para la detección de armas pueden mejorar significativamente la seguridad ciudadana, su implementación debe ser acompañada de un análisis ético y legal riguroso. Solo así se garantizará que estos avances tecnológicos contribuyan verdaderamente al bienestar colectivo sin vulnerar derechos fundamentales.

Capítulo II

2. Análisis y Diseño

2.1. Introducción al análisis del problema

La violencia armada creció en Ecuador y los comercios pequeños quedaron entre los blancos más expuestos, como documenta el Capítulo I. Frente a esa presión, la visión por computadora aporta algo que el circuito cerrado tradicional no da: un modelo que vigila el flujo de video sin descanso y avisa cuando aparece un arma. Este capítulo analiza y diseña ese sistema, y arranca por el problema tal como se presenta dentro del local.

2.1.1. Motivaciones prácticas para el desarrollo del sistema

Tres conversaciones dieron origen al sistema. El autor habló con un miembro de las Fuerzas Armadas, con una guardia penitenciaria y con un comerciante del sector privado, personas que trabajan a diario en entornos expuestos a delitos con armas. Los tres coincidieron en el mismo punto de partida: los lugares donde se concentra gente —supermercados, terminales, centros educativos— dependen de cámaras que nadie mira en el momento en que ocurre el hecho.

De esa coincidencia salió el requisito central: que el aviso salga solo, en cuanto la cámara ve el arma, y alcance a quien puede reaccionar antes de que el hecho se consume. Un aviso así acorta el tiempo de reacción y entrega evidencia visual desde el primer segundo. Esa demanda, planteada por gente que trabaja en riesgo, es la que el proyecto atiende.

2.1.2. Contexto de aplicación en seguridad ciudadana

Detrás del sistema hay una restricción concreta. En Ecuador, como en buena parte de la región, las instituciones policiales trabajan con presupuestos, personal y equipamiento cortos. Donde no hay gente suficiente para mirar cada cámara, un modelo que observa el flujo de video sin fatiga complementa la vigilancia tradicional en lugar de sustituirla.

El objetivo es directo. El sistema detecta armas en tiempo real sobre el flujo de las cámaras de videovigilancia ya instaladas, con un modelo entrenado para esa clase de objeto. El mismo esquema es aplicable a supermercados, recintos penitenciarios, instalaciones militares, escuelas o eventos masivos: cambia el escenario, no la arquitectura. El diseño admite ese escalamiento siempre dentro de los límites legales del monitoreo y con respeto a los derechos fundamentales, en los términos revisados en la sección 1.6.

2.2. Metodología empleada: CRISP-DM

El desarrollo siguió CRISP-DM (Cross Industry Standard Process for Data Mining). La razón es de fondo. Este no es un proyecto de software convencional, donde los requisitos se fijan y luego se programan, sino un proceso analítico que arranca en la comprensión de los datos y termina en la validación del modelo en escenarios reales. CRISP-DM cubre ese recorrido y, por ser iterativo, admite volver atrás cuando la evaluación lo exige.

2.2.1. Justificación del uso de CRISP-DM en proyectos de IA

Las metodologías tradicionales de ingeniería de software dan por sentado que el comportamiento del sistema se especifica antes de construirlo. En aprendizaje automático esa premisa falla: el comportamiento sale de los datos, y el ciclo real consiste en entrenar, medir, ajustar y volver a entrenar. CRISP-DM encadena esas etapas sin forzarlas —del entendimiento del problema a la preparación de datos, de ahí al modelado con YOLOv8 y a la evaluación continua—, y por eso guio este proyecto.

Hay un motivo adicional. Reconocer armas en entornos visuales variados es un problema experimental: ningún resultado de entrenamiento garantiza el comportamiento en campo, y el marco metodológico debe asumir la iteración como regla y no como excepción.

2.2.2. Fases del modelo aplicadas al sistema de detección

CRISP-DM consta de seis fases. La Tabla 11 muestra cómo se adaptó cada una al desarrollo del sistema de detección de armas.

Tabla 11:

Aplicación de las fases CRISP-DM al sistema propuesto

Fase CRISP-DM	Aplicación en el proyecto
Comprensión del negocio	Identificación del problema con base en testimonios reales (militar, penitenciario, comercial) y objetivos de seguridad ciudadana.
Comprensión de los datos	Análisis de datasets existentes con imágenes etiquetadas de armas, evaluación de su calidad y distribución.
Preparación de los datos	Limpieza, etiquetado, redimensionamiento, y técnicas de aumento de datos para robustecer el modelo de detección.
Modelado	Selección del modelo YOLOv8, ajustes de hiperparámetros, y entrenamiento sobre el dataset procesado.
Evaluación	Cálculo de métricas como precisión, recall y mAP en escenarios simulados. Evaluación con umbrales ajustables.
Despliegue	Integración del modelo en el sistema cliente con interfaz PyQt5, generación de alertas y visualización en panel web.

Nota. Elaboración propia.

Esta estructura ordenó el trabajo en etapas sucesivas, desde la definición del problema hasta la entrega de una solución funcional validada en campo. El carácter iterativo del modelo se notó sobre todo durante la etapa de pruebas. Cada ciclo dejó ajustar el sistema según lo observado en la ronda anterior y mejorar su desempeño en condiciones reales de operación, de modo que ganara confiabilidad antes de pasar al uso diario.

2.3. Levantamiento de requerimientos

La fase de levantamiento de requerimientos constituye una etapa clave dentro del análisis, pues define las funcionalidades esenciales que debe cubrir el sistema para

responder eficazmente a escenarios de riesgo reales. A diferencia de enfoques basados en entrevistas formales o talleres institucionales, en este caso se recurrió a fuentes informales y testimonios personales confiables, que permitieron obtener una perspectiva directa de las necesidades prácticas de seguridad.

2.3.1. Fuentes informales de recolección (casos reales)

El autor levantó los requerimientos en conversaciones informales con tres personas de sectores distintos, todas expuestas a delitos con armas. Un miembro activo de las Fuerzas Armadas señaló la necesidad de detectar armas incluso en áreas restringidas y con personal autorizado presente. Un guardia penitenciario habló de otra urgencia: frenar el ingreso de armas a los centros carcelarios, donde la violencia se ha intensificado. El tercero, comerciante y propietario de un supermercado, describió cómo la percepción de inseguridad ahuyenta a los clientes y propuso una alerta automática ante comportamientos sospechosos.

De esas experiencias salieron tres requerimientos concretos: operación en tiempo real, alertas inmediatas con evidencia visual adjunta y un filtrado que evite saturar al operador con notificaciones repetidas. El instrumento estructurado que se aplicó a los propietarios de los dos establecimientos, ya con el sistema en operación, corresponde a una etapa distinta y se detalla en el apartado 3.6.3 y en el Anexo H.

2.4. Requisitos del sistema

El trabajo fijó por escrito, antes de diseñar, qué debía hacer el sistema y con qué calidad. Esta sección recorre esos requisitos —funcionales y no funcionales—, las interfaces y las reglas de negocio que condicionan la operación.

2.4.1. Requisitos funcionales y no funcionales

La distinción es la habitual en ingeniería de software: los funcionales describen comportamiento —qué hace el sistema ante qué entrada— y los no funcionales fijan criterios de rendimiento, seguridad y mantenibilidad.

En la Tabla 12 se clasifica por categoría y prioridad estosrequisitos que fueron definidos a partir de escenarios reales y necesidades prácticas consultadas en entornos militares y comerciales, como se analizó en secciones anteriores.

Tabla 12:

Requisitos funcionales y no funcionales del sistema.

Código	Descripción	Prioridad
RF-DET-01	Detección en tiempo real de armas con modelo YOLO	Alta
RF-ALT-01	Envío automático de alertas con imagen, timestamp y destinatario	Alta
RF-ALM-01	Almacenamiento estructurado de alertas en base de datos	Alta
RF-VIS-01	Visualización ordenada de alertas y detalles relevantes	Alta
RF-AUT-01	Autenticación y gestión de usuarios mediante tokens	Alta
RF-MON-01	Monitoreo en tiempo real del uso de CPU, memoria y GPU	Media
RNF-PER-01	Procesamiento en tiempo real con latencia máxima de 30 segundos	Alta
RNF-SEG-01	Comunicación cifrada por HTTPS y almacenamiento seguro de contraseñas	Alta
RNF-FIA-01	Alta disponibilidad del sistema (99,9 %) y sistema de logs	Alta
RNF-MAN-01	Código documentado y pruebas unitarias implementadas	Media

Nota. Elaboración propia. RF = requisito funcional; RNF = requisito no funcional; CPU = unidad central de procesamiento; GPU = unidad de procesamiento gráfico; HTTPS = protocolo seguro de transferencia de hipertexto.

2.4.2. Interfaces del sistema (usuario, hardware, software, comunicación)

El sistema cuenta con diversas interfaces que permiten la interacción fluida entre sus componentes y con los usuarios. La Tabla 13 resume las interfaces identificadas para la implementación.

Tabla 13:

Interfaces del sistema

Tipo de interfaz	Descripción
Usuario (RF-UI)	Interfaces PyQt5 para video y detección; interfaz web para visualización, login y alertas
Hardware (RF-HW)	Compatibilidad con cámaras IP sobre RTSP (Tapo, Hikvision, Dahua, Reolink o URL manual) y con cámaras web USB; ejecución en hardware estándar, con o sin GPU
Software (RF-SW)	Cliente en Python, servidor en Django, comunicación mediante API REST
Comunicación (RF-COM)	Protocolo HTTPS cifrado, autenticación basada en tokens para API REST

Nota. Elaboración propia. RF = requisito funcional; UI = interfaz de usuario; RTSP = protocolo de transmisión en tiempo real; URL = localizador uniforme de recursos; USB = bus serie universal; API REST = interfaz de programación de aplicaciones de transferencia de estado representacional; HTTPS = protocolo seguro de transferencia de hipertexto.

Estas interfaces aseguran que el sistema sea accesible, interoperable y seguro durante su operación.

2.4.3. Reglas de negocio y condiciones de operación

Las reglas de negocio definen la lógica que rige el comportamiento del sistema frente a eventos críticos como la detección de armas. Estas reglas se complementan con condiciones técnicas necesarias para su funcionamiento adecuado:

a. Reglas de negocio:

- Cuando se detecte un arma, se debe capturar una imagen, generar una alerta y enviarla automáticamente.
- El servidor debe validar que el destinatario sea una dirección de correo electrónico y despachar la alerta por ese canal.
- Las alertas deben registrarse con identificador único, imagen, ubicación, usuario, fecha y hora.

b. Condiciones de operación:

- El sistema debe operar conectado a internet estable.
- Requiere compatibilidad con cámaras IP sobre RTSP y con cámaras web USB.
- El servidor debe estar disponible 24/7 con mantenimiento planificado menor a 2 horas/mes.

Estas condiciones son esenciales para garantizar la disponibilidad del servicio, la respuesta oportuna ante eventos críticos y la integridad de la información.

2.5. Modelo de casos de uso

2.5.1. Descripción de actores y funcionalidades principales

En el modelo de casos de uso intervienen dos clases de actor. Los humanos —el guardia, el administrador— ingresan al sistema, consultan las alertas y gestionan usuarios; los automatizados son módulos internos que disparan el flujo sin intervención de nadie: el detector que analiza cada cuadro, el módulo que arma y envía la notificación. Distinguirlos importa porque el comportamiento esperado se rastrea de forma distinta según quién inicia la acción: un caso de uso humano se valida en interfaz, uno automatizado se valida en los logs del backend.

La Tabla 14 recoge cada actor con sus funciones.

Tabla 14:

Actores principales del sistema y sus funcionalidades

Actor	Tipo	Funcionalidades principales
Nuevo Usuario	Humano	Registro inicial vía interfaz web, validación de credenciales y creación de cuenta
Usuario del Sistema	Humano	Inicio de sesión en PyQt5 y dashboard web, monitoreo en tiempo real, configuración de parámetros y consultas históricas
Sistema de Detección	Automatizado	Análisis de video con YOLOv8, detección de armas, captura de imagen, generación y almacenamiento de alertas
Sistema de Notificaciones	Automatizado	Validación del correo del receptor, envío automático de la alerta por correo electrónico, registro de envíos

Nota. Elaboración propia.

Este enfoque estructurado permite comprender claramente las interacciones que cada actor tiene con el sistema. Por ejemplo, el Usuario del Sistema opera tanto la aplicación cliente como el panel web, garantizando una supervisión completa del entorno. Mientras tanto, los actores automatizados, como el Sistema de Detección y de Notificaciones, operan sin intervención humana para asegurar la eficiencia del flujo de trabajo, respondiendo en casi tiempo real ante eventos críticos.

Esta distinción es fundamental para el posterior desarrollo de los diagramas de casos de uso, ya que permitirá visualizar claramente los límites del sistema y los puntos de interacción relevantes.

2.5.2. Diagramas UML de casos de uso generales y específicos

a. Diagrama General

A partir de los actores identificados y sus funcionalidades, se elaboró un diagrama general de casos de uso en UML que reúne las interacciones del sistema. Agrupa los

módulos en bloques lógicos, como Detección Automática, Gestión de Usuarios o Consulta de Alertas, entre otros, y relaciona los actores humanos con los automatizados.

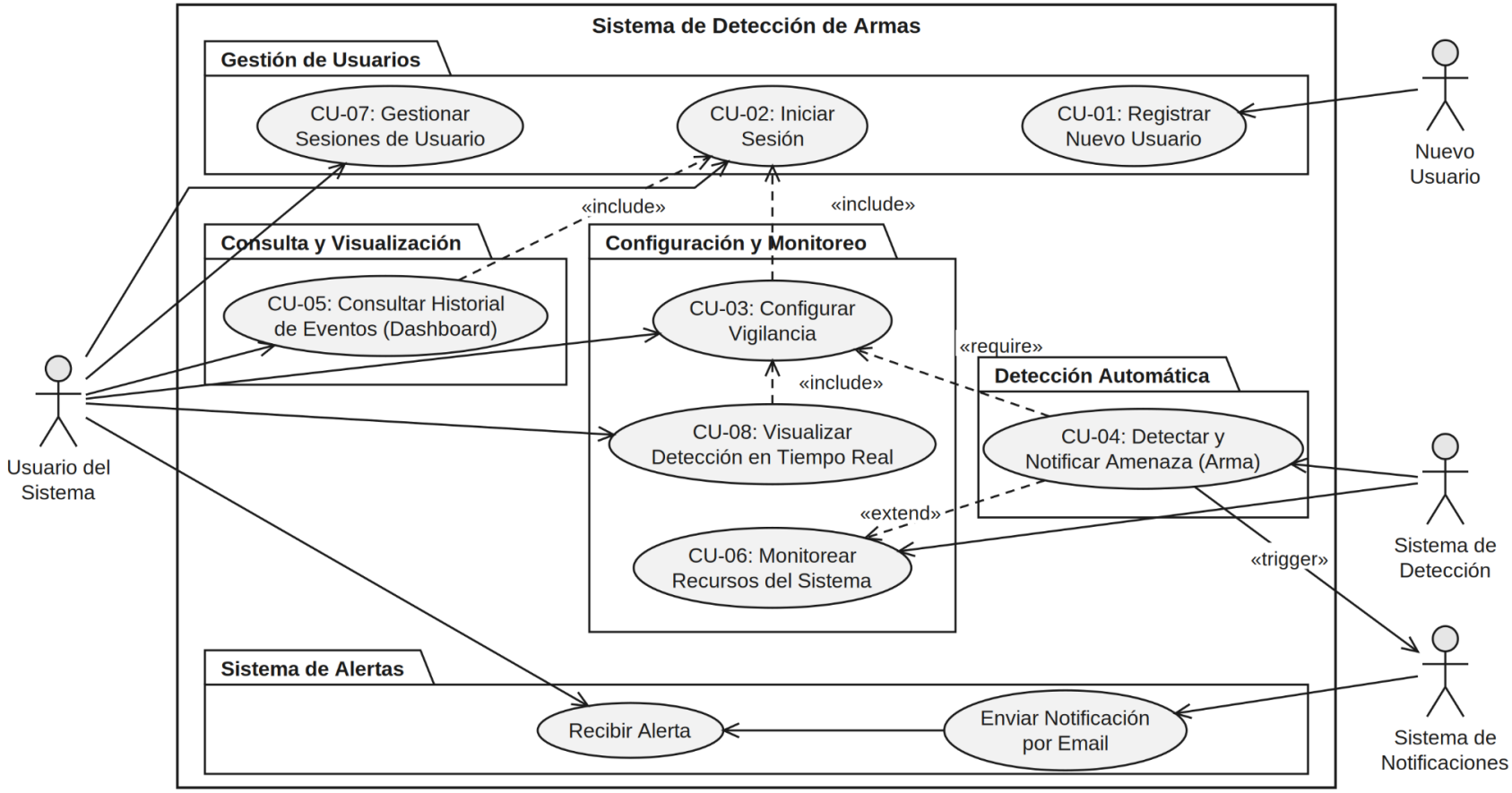
La Figura 7 muestra el diagrama con los principales procesos del flujo operativo:

El actor Nuevo Usuario solo se registra en la plataforma. El Usuario del Sistema, ya autenticado, opera desde la aplicación cliente o el panel web: configura los parámetros de vigilancia (CU-03), observa la detección en tiempo real (CU-08), consulta el historial de eventos (CU-05) y monitorea el uso de recursos (CU-06).

Los actores automatizados cumplen funciones internas: el Sistema de Detección ejecuta el análisis de video (CU-04) y genera alertas estructuradas; el Sistema de Notificaciones valida el correo del destinatario y remite la alerta.

Figura 7:

Diagrama general de casos de uso del sistema de detección de armas



Nota. Elaboración propia.

b. Diagrama Específicos

Módulo de Autenticación:

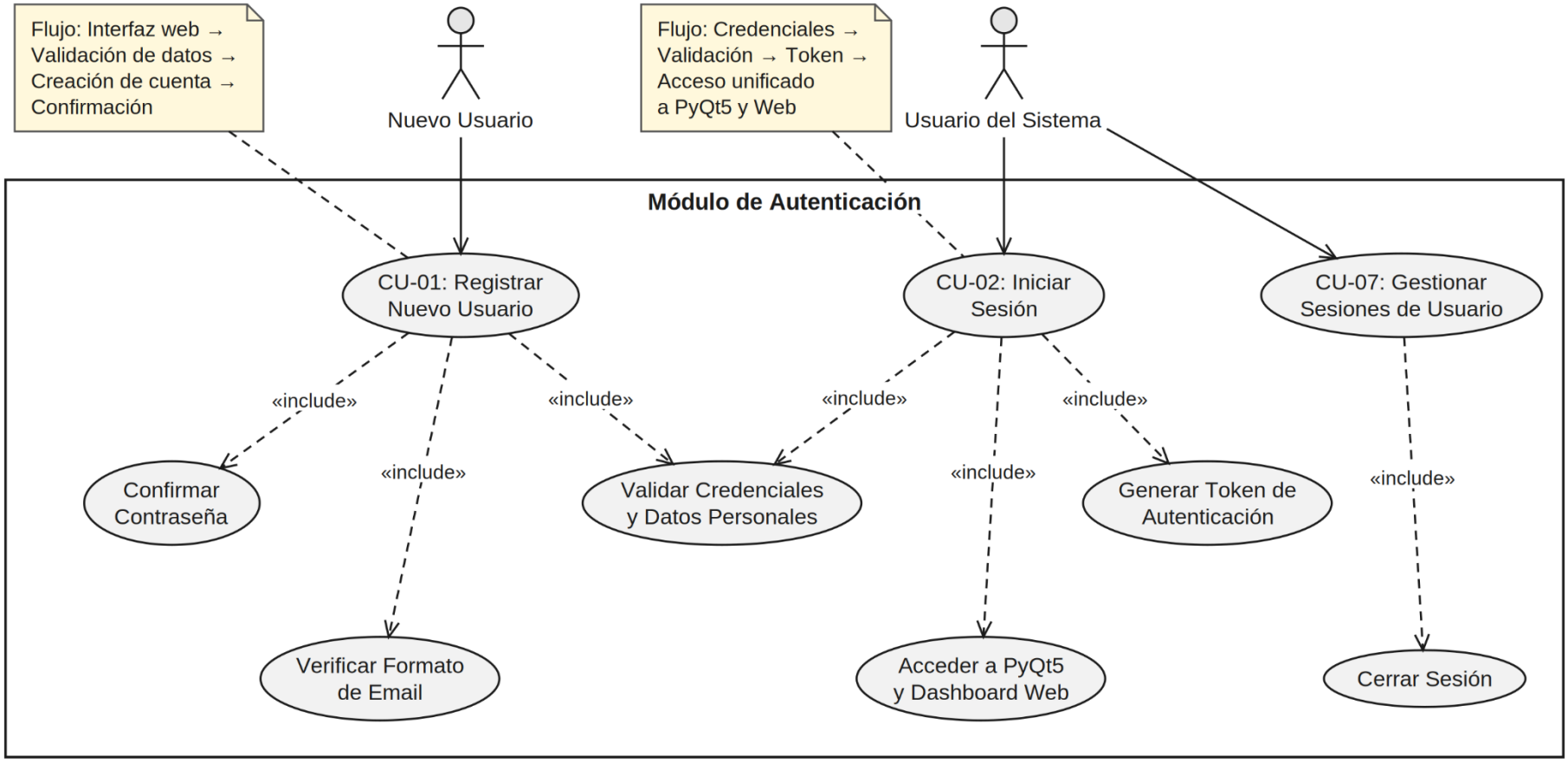
La Figura 8 detalla el módulo de autenticación, puerta de entrada al sistema tanto desde el cliente PyQt5 como desde el panel web. El actor Nuevo Usuario ejecuta CU-01 (Registrar Usuario). El Usuario del Sistema inicia sesión con CU-02 y administra sus sesiones y su token de autenticación mediante CU-07. Las subfunciones son controles concretos: el sistema valida el formato del correo, verifica la contraseña y mantiene un acceso unificado, de modo que nadie carga credenciales separadas por interfaz.

Módulo de Detección y Alerta: el núcleo del sistema está en la Figura 9. El actor automatizado Sistema de Detección ejecuta CU-04 (Detectar y Notificar Amenaza), que encadena cuatro operaciones. Analiza el flujo de video, descarta toda detección por debajo de 0,75 de confianza, captura la evidencia fotográfica y arma la alerta estructurada. El Sistema de Notificaciones recibe esa alerta, valida el correo del receptor y despacha el mensaje. Aquí cada milisegundo cuenta: toda subfunción del módulo corre dentro de la ventana de tiempo real, y es sobre esta cadena donde la investigación mide los tramos T1 a T4.

Módulo de Consulta y Visualización: Queda el trabajo posterior al incidente. En la Figura 10, el Usuario del Sistema accede desde el dashboard web al historial de eventos mediante CU-05 (Consultar Historial de Eventos), que se extiende en visualizar las imágenes capturadas, filtrar por fecha, descripción, receptor o usuario, y abrir el detalle de cada evento. Sin este módulo cada alerta sería un hecho aislado; con él, el operador reconstruye qué pasó, cuándo y con qué evidencia.

Figura 8:

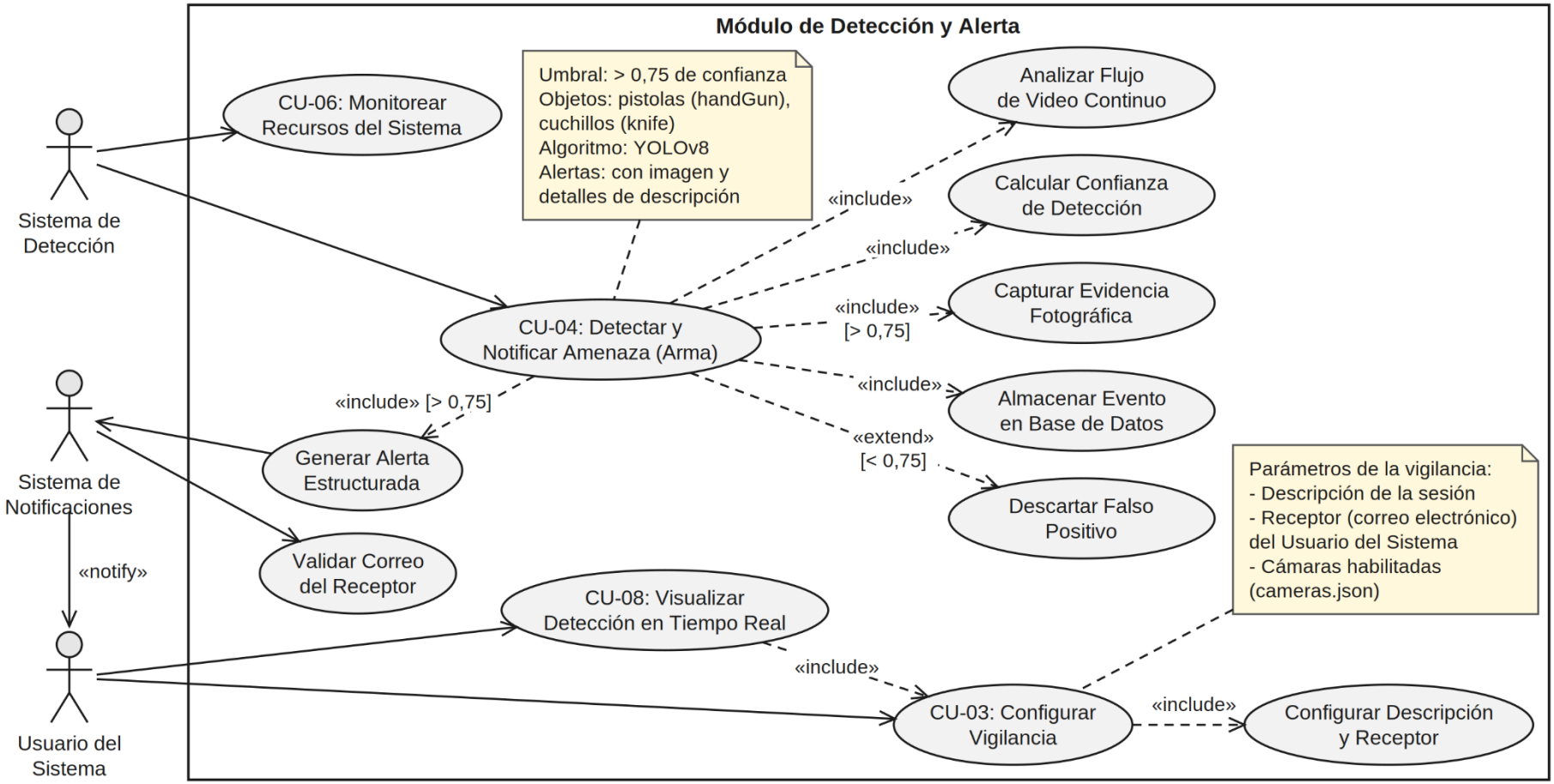
Diagrama de casos de uso - Módulo de Autenticación



Nota. Elaboración propia.

Figura 9:

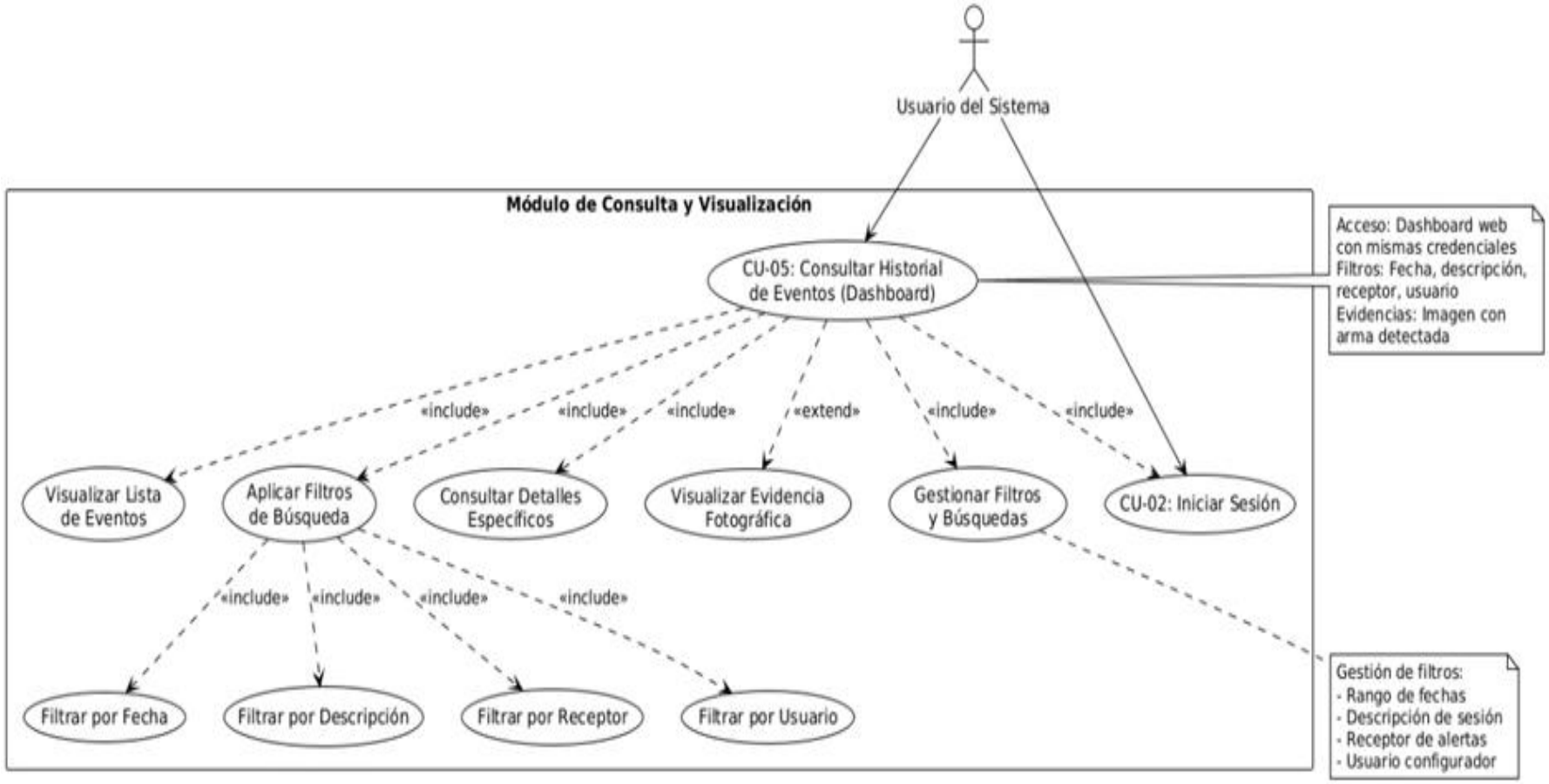
Casos de uso, Modulo de Detección y Alerta.



Nota. Elaboración propia.

Figura 10:

Diagrama de casos de uso - Módulo de Consulta y Visualización



Nota. Elaboración propia.

2.6. Diagramas de análisis

2.6.1. Diagrama de clases principal

El diagrama de clases constituye una representación estructural de los componentes clave del sistema, permitiendo visualizar la relación entre entidades tanto en el cliente (PyQt5) como en el servidor (Django). Este modelo orientado a objetos establece la base para la implementación funcional de la solución, describiendo atributos, métodos y vínculos lógicos entre clases. En la Figura 11 se presenta el diagrama de clases principal del sistema de detección de armas, el cual está dividido en dos grandes módulos: Cliente (PyQt5) y Servidor (Django).

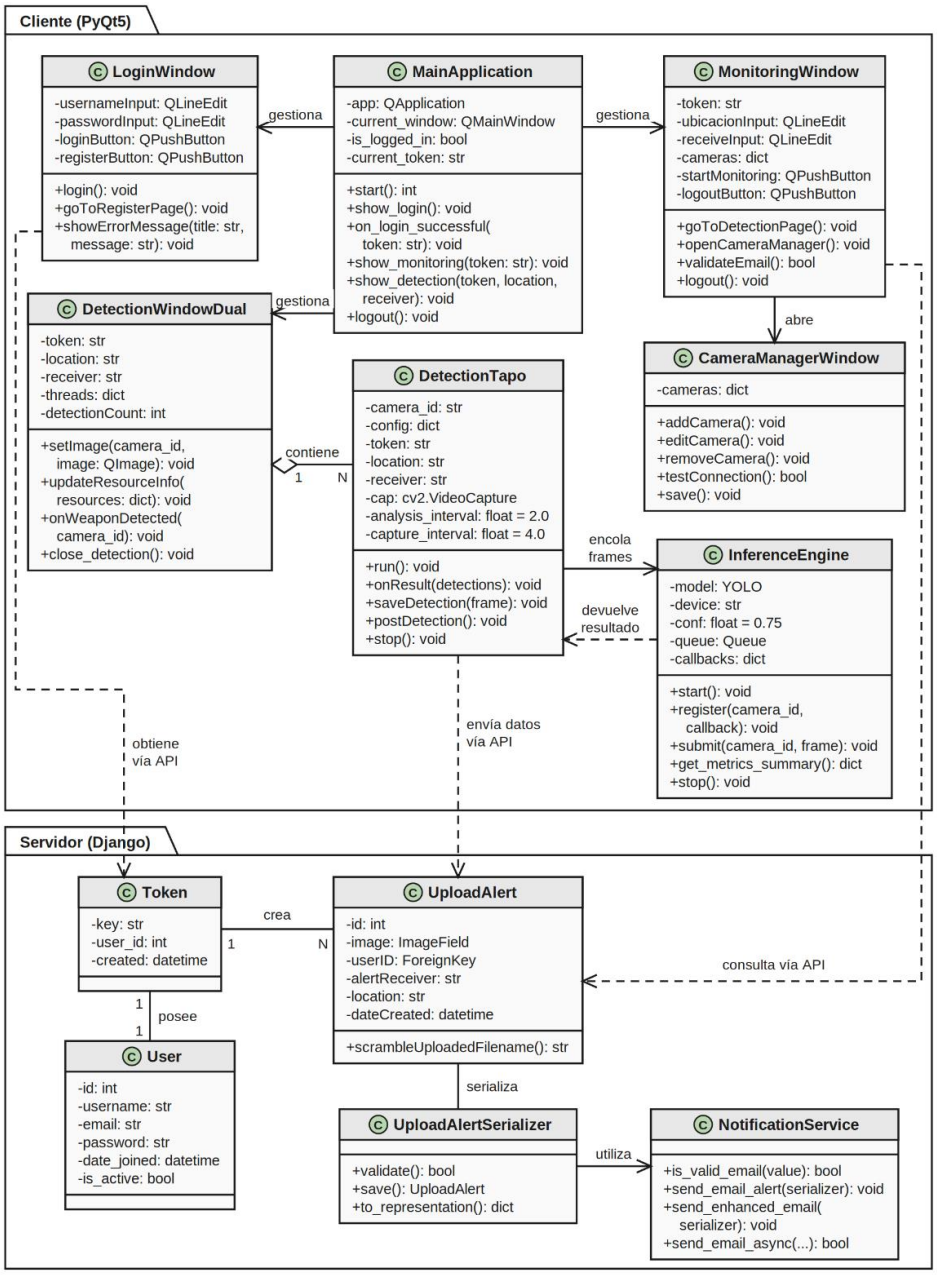
Del lado del cliente, la clase MainApplication hace de núcleo de la interfaz gráfica: gobierna la navegación entre ventanas y el control de sesión. Interactúa con LoginWindow, encargada de autenticar al usuario mediante sus credenciales, y con DetectionWindow y MonitoringWindow, que ejecutan la lógica de detección en tiempo real y el monitoreo de recursos, respectivamente. Estas clases se apoyan en dos piezas. DetectionTapo levanta un hilo por cámara y se encarga de la captura del flujo de video (cv2.VideoCapture sobre RTSP o webcam); InferenceEngine carga el modelo YOLO una sola vez y lo comparte entre todas las cámaras activas, devuelve cada resultado a la cámara que lo originó y envía la evidencia al servidor por la API REST.

En el backend (servidor Django), el modelo User administra la información básica de los usuarios, mientras que Token permite el manejo de sesiones y autenticación vía API. Cuando se detecta un arma, los datos son enviados al servidor, que los procesa y guarda mediante la clase UploadAlert, incluyendo la imagen, el receptor y la localización. Esta clase se apoya en UploadAlertSerializer para la serialización/deserialización de los datos. Finalmente, el NotificationService despacha la alerta al destinatario por correo electrónico mediante la API HTTP de Brevo, en un hilo secundario que no bloquea la respuesta al cliente.

Los módulos se comunican por una API REST protegida: DetectionWindow y MonitoringWindow envían información estructurada, y el servidor la almacena y la procesa. El ciclo completo va de la autenticación al monitoreo, la detección, la alerta y la notificación.

Figura 11:

Diagrama de clases del sistema de detección de armas con arquitectura cliente-servidor



Nota. Elaboración propia.

2.6.2. Diagrama de secuencia: proceso de detección y alerta

La detección de armas en tiempo real es el núcleo operativo del sistema. El ciclo empieza cuando el usuario se autentica y termina cuando consulta la alerta ya registrada, después de que el sistema la genera, la transmite y la notifica. La Figura 12 traza ese recorrido con el diagrama de secuencia e identifica cada intercambio entre el cliente PyQt5, el modelo YOLO, el servidor Django, la base de datos y el sistema de notificaciones.

a. Autenticación y configuración inicial

El usuario ingresa sus credenciales en la interfaz PyQt5 y el cliente envía una solicitud POST al endpoint /api/get_auth_token/. Si las credenciales son válidas, el servidor devuelve un token de autenticación y abre el acceso al sistema. El usuario configura entonces los parámetros de monitoreo —ubicación y destinatario— y arranca el hilo de detección con los argumentos necesarios (token, location, receiver); ese hilo carga el modelo YOLOv8 entrenado y activa la cámara.

b. Proceso continuo de detección

Cada frame capturado es procesado por el hilo de detección (Detection Thread). Si se detecta un objeto clasificado como handGun o knife con un nivel de confianza mayor al umbral configurado (0,75), el sistema dibuja un bounding box sobre el arma detectada y actualiza la interfaz del cliente. Para evitar alertas repetidas, el sistema evalúa si han transcurrido al menos 4 segundos desde la última captura válida.

c. Captura y transmisión de evidencia

Cumplido el intervalo, el frame es guardado localmente como imagen y se ejecuta un POST al endpoint /api/images/ incluyendo:

- El archivo de imagen capturado.
- Metadatos (ubicación, receptor, timestamp).
- Token de autenticación en la cabecera.

Este conjunto de datos representa la transición al estado de alerta transmitida.

d. Procesamiento del servidor y generación de notificaciones

El servidor Django recibe los datos, valida la imagen, la estructura y la autenticidad del token mediante el UploadAlertSerializer, y persiste los datos creando un nuevo objeto UploadAlert. El campo alertReceiver se contrasta con una expresión regular de correo electrónico; si el formato es válido, se dispara la notificación:

- Se envía un correo electrónico mediante la API HTTP de Brevo, en un hilo secundario que no bloquea la respuesta al cliente.

e. Registro y persistencia de la alerta

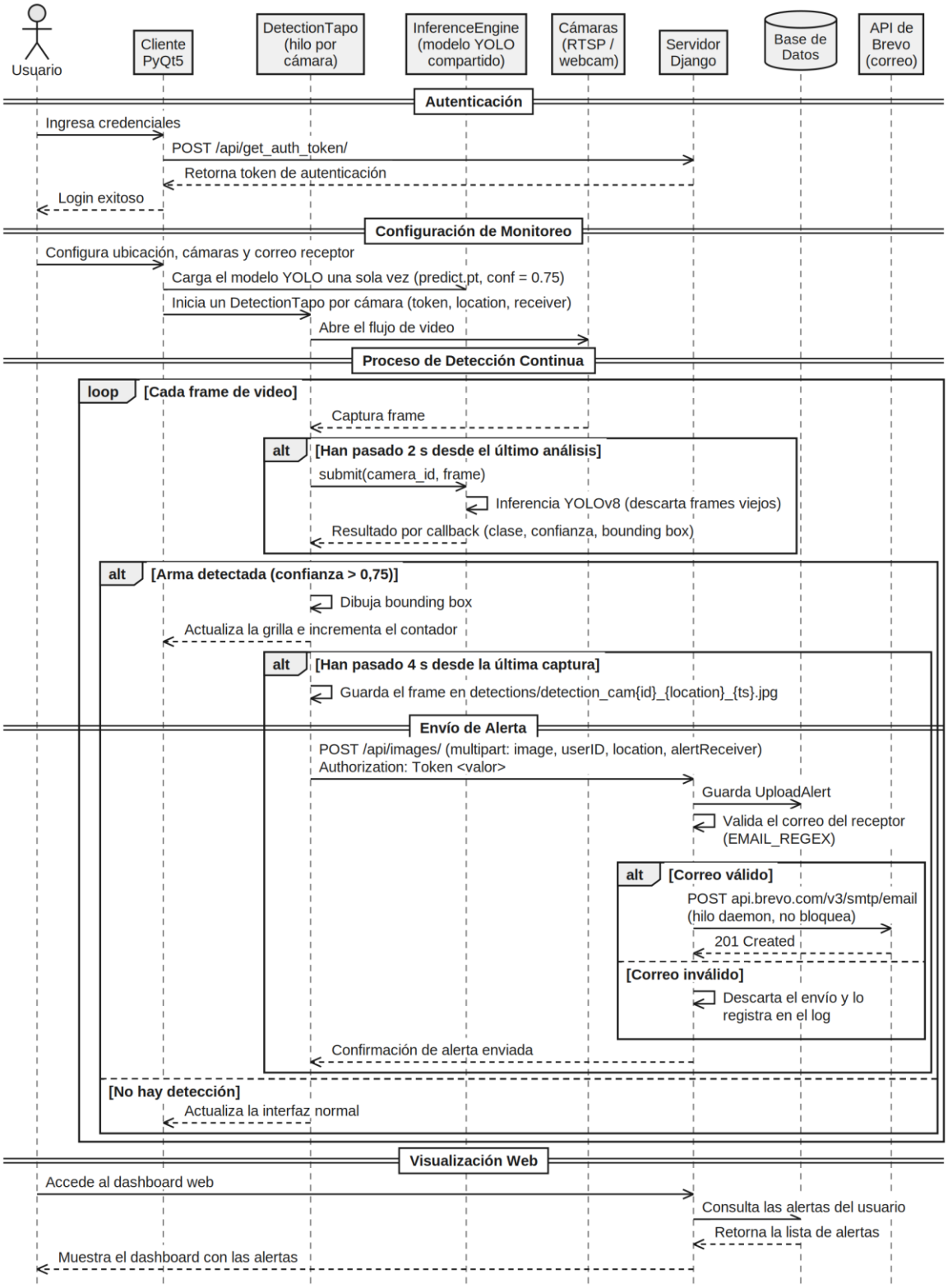
El servidor guarda los datos en la base de datos PostgreSQL, y la imagen es almacenada en la nube mediante Amazon S3 con nombre ofuscado generado por scrambleUploadedFilename (). Esto marca el estado de alerta registrada.

f. Consulta y visualización por el usuario

El usuario entra al dashboard web, se autentica y revisa todas sus alertas históricas. El backend Django atiende las consultas y django-filters las acota por fecha, ubicación o receptor. Cada alerta aparece con su imagen —servida desde S3— y los metadatos del evento.

Figura 12:

Diagrama de secuencia del proceso de detección y alerta en tiempo real.



Nota. Elaboración propia.

2.6.3. Diagrama de estados: ciclo de vida de una alerta

Una alerta atraviesa una secuencia fija de estados, y esa secuencia deja trazado el incidente desde que se origina hasta que alguien lo consulta. El diagrama de estados dibuja ese recorrido: qué evento dispara cada transición y qué hace el sistema en cada etapa.

La Figura 13 recorre el ciclo completo, desde que el modelo YOLOv8 detecta el arma hasta que el usuario abre la alerta en el dashboard web.

a. Estado inicial: Detectada

El ciclo arranca cuando el modelo YOLOv8 clasifica un objeto como handGun o knife con una confianza superior al umbral establecido (0,75). En ese primer estado el sistema registra el frame, dibuja el bounding box sobre el objeto y marca un timestamp. El evento activa la alerta y abre la transición siguiente.

b. Transición a Capturada

El sistema comprueba que hayan pasado al menos cuatro segundos desde la última alerta y guarda el frame en la carpeta detections/ con el nombre detection_cam{camera_id}_{location}_{AAAAMMDD-HHMMSS}.jpg. La alerta queda en estado Capturada.

c. Estado Transmitida

El sistema avanza al estado Transmitida cuando realiza una petición POST segura al endpoint /api/images/, transmitiendo:

- La imagen capturada.
- Los metadatos (ubicación, receptor).
- El token de autenticación.

Con ese conjunto el servidor Django tiene todo lo que necesita para procesar la alerta.

d. Estado Procesada

Una vez en el servidor, la alerta entra al estado Procesada, donde se ejecuta la validación de datos mediante UploadAlertSerializer y se valida el formato del receptor (correo electrónico) mediante expresiones regulares.

e. Estados paralelos: Notificada y Registrada

Desde Procesada, el flujo se bifurca:

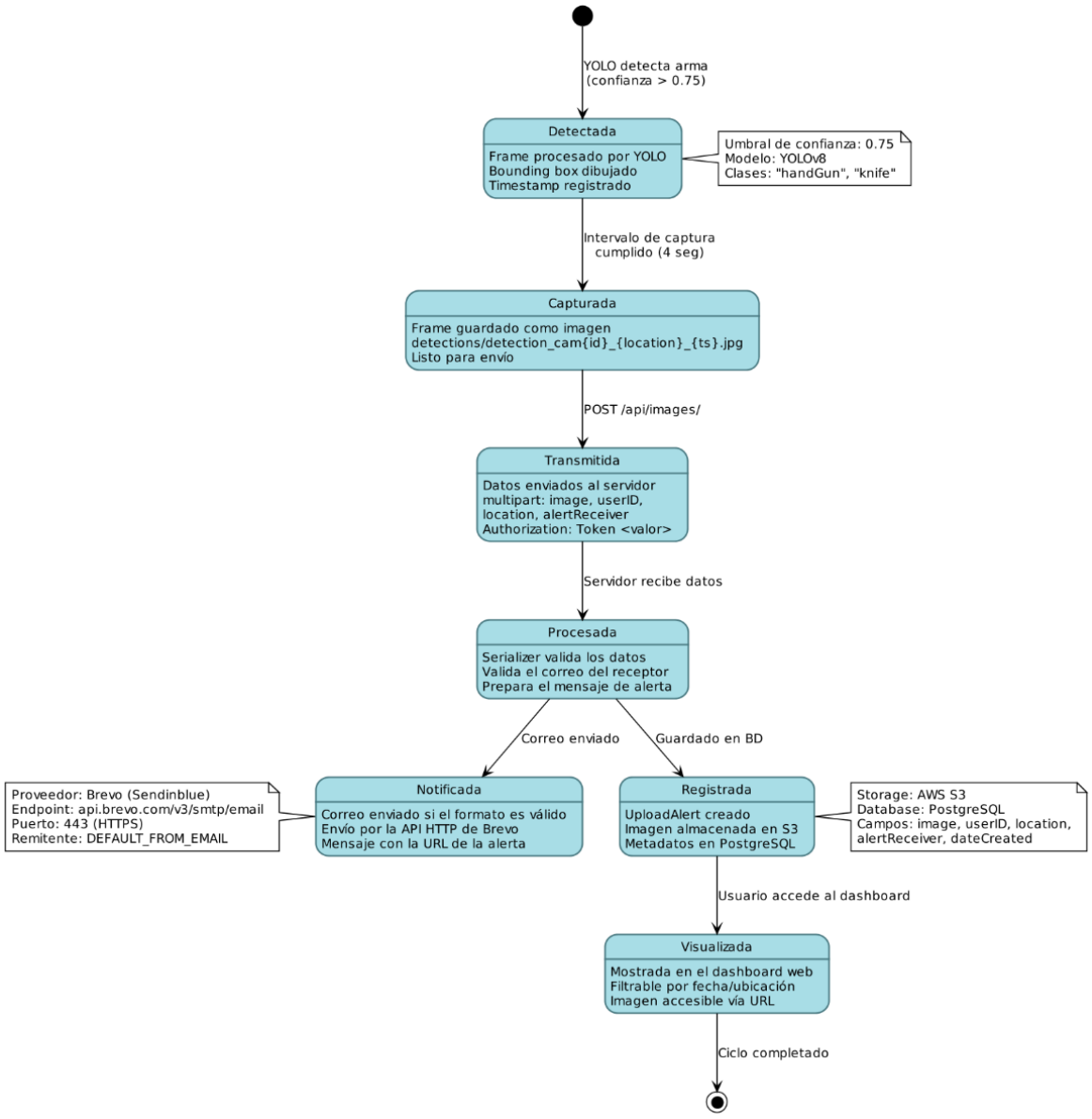
- En el estado Notificada, se envía el mensaje correspondiente:
  - Por correo electrónico, mediante la API HTTP de Brevo (https://api.brevo.com/v3/smtp/email), con cuerpo HTML y respaldo en texto plano.
- En Registrada, los datos son almacenados en la base de datos PostgreSQL y la imagen es subida a AWS S3 con nombre ofuscado generado por scrambleUploadedFilename(). Se crean así registros persistentes con los campos image, userID, location, alertReceiver y dateCreated. El nivel de confianza de la detección no se persiste en esta versión del sistema.

f. Estado final: Visualizada

Finalmente, la alerta es Visualizada cuando el usuario accede al dashboard web, donde puede consultar sus alertas filtradas por token, fecha o ubicación. La evidencia gráfica (imagen) se carga desde Amazon S3 y se acompaña de los metadatos correspondientes. Esta etapa marca la finalización del ciclo de vida de una alerta y su disponibilidad para análisis retrospectivo.

Figura 13:

Diagrama de estados del ciclo de vida de una alerta de detección de armas



Nota. Elaboración propia.

2.7. Diseño de la arquitectura del sistema

El diseño arquitectónico del sistema de detección de armas responde a un enfoque distribuido, modular y escalable, basado en el modelo cliente-servidor. La solución combina procesamiento local de video en tiempo real con servicios centralizados de almacenamiento, autenticación, gestión de alertas y notificaciones, lo que permite una operación eficiente, segura y trazable.

2.7.1. Arquitectura cliente-servidor y flujo de información

El sistema se estructura en tres capas principales: cliente (PyQt5), servidor (Django) y servicios externos. Esta separación permite garantizar la independencia de responsabilidades, facilidad de mantenimiento y posibilidad de escalamiento horizontal o vertical. En la Figura 14 se presenta el esquema general de la arquitectura, en el que se distinguen claramente los componentes de software y sus interacciones a través de canales seguros (HTTPS + token de autenticación de Django REST Framework (DRF)).

a. Capa Cliente (PyQt5)

Esta capa actúa como interfaz de interacción con el usuario y motor de procesamiento local. Está compuesta por los siguientes módulos:

- **MainApplication:** Controlador principal de la aplicación.
- **LoginWindow:** Permite la autenticación inicial mediante comunicación con el servidor vía API REST.
- **MonitoringWindow:** Permite configurar la ubicación y receptor de alertas.
- **DetectionWindowDual:** Visualiza en una grilla dinámica el video en vivo de todas las cámaras habilitadas, con las detecciones activas, los FPS por cámara y el estado del sistema.
- **InferenceEngine y DetectionTapo** El primero carga el modelo YOLOv8 una sola vez y lo comparte entre todas las cámaras; el segundo levanta un hilo por cámara que captura los frames y recibe de vuelta el resultado de la inferencia.

b. Capa Servidor (Django)

El servidor centraliza la lógica del sistema. Está compuesto por:

- **API REST:** Gestiona endpoints para autenticación, subida de alertas y gestión de tokens.
- **Modelos Django:** Representan usuarios, alertas y tokens.

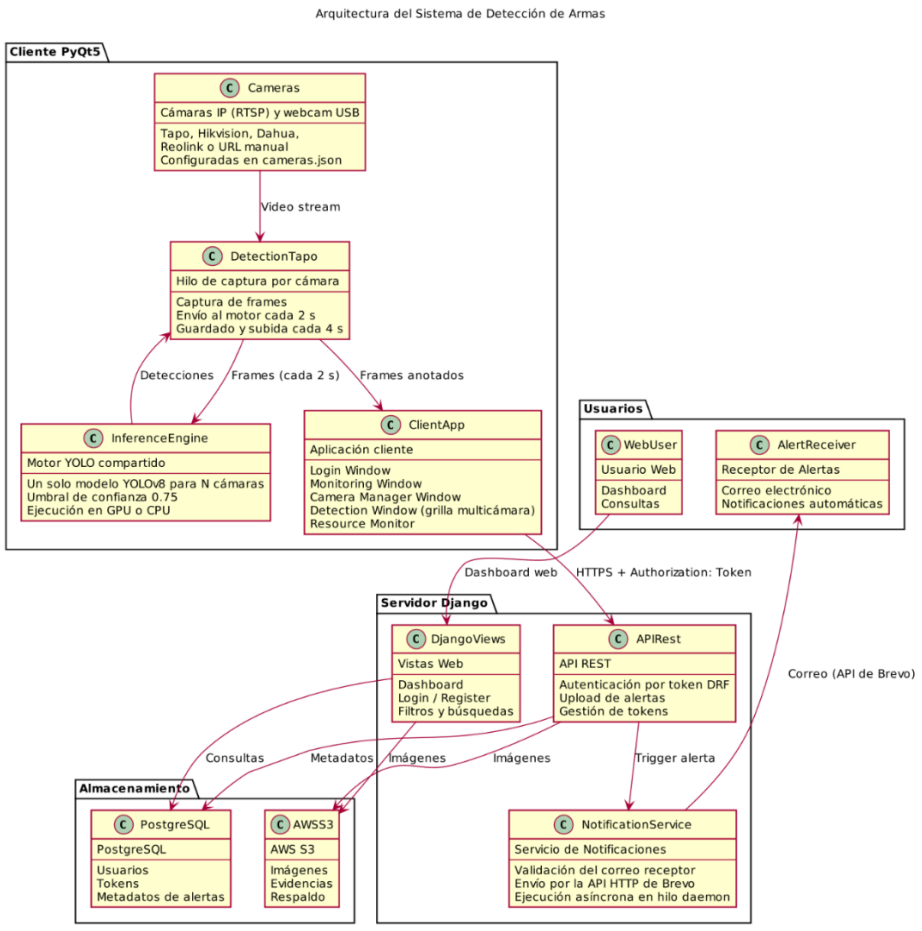
- **Vistas Web:** Permiten la visualización del dashboard, login web y filtros de consulta.
- **NotificationService:** Módulo que valida el correo del receptor y ejecuta el envío del correo de alerta.

c. Servicios Externos

- **PostgreSQL:** Almacena usuarios, tokens y metadatos de alertas.
- **AWS S3:** Guarda imágenes capturadas como evidencia.
- **API de Brevo (Sendinblue):** Se utiliza para el envío automático de las notificaciones por correo electrónico.

Figura 14:

Arquitectura cliente-servidor del sistema de detección de armas



Nota. Elaboración propia

d. Flujo de información del sistema

La Figura 15 representa el flujo completo de información del sistema, desde la detección inicial hasta la notificación y almacenamiento final.

e. Captura y procesamiento inicial

El proceso comienza con la captura de video desde una o varias cámaras —IP sobre RTSP o webcam USB— declaradas en el archivo cameras. json del cliente PyQt5. Cada frame es analizado por el modelo YOLOv8 para identificar objetos sospechosos. Si se detecta un arma con un nivel de confianza superior a 0,75, se activa el proceso de alerta.

f. Validación y preparación de datos

El frame detectado se guarda como imagen local. Se preparan los datos, incluyendo los metadatos (usuario, ubicación, receptor) y el token de autenticación.

g. Transmisión segura al servidor

El cliente realiza una solicitud POST al endpoint /api/images/ del servidor Django, enviando los datos mediante protocolo HTTPS.

h. Procesamiento y almacenamiento distribuido

El servidor:

- Guarda los metadatos en PostgreSQL.
- Sube la imagen capturada a AWS S3.
- Valida que el receptor sea una dirección de correo electrónico.

i. Envío de notificaciones

Según el receptor detectado:

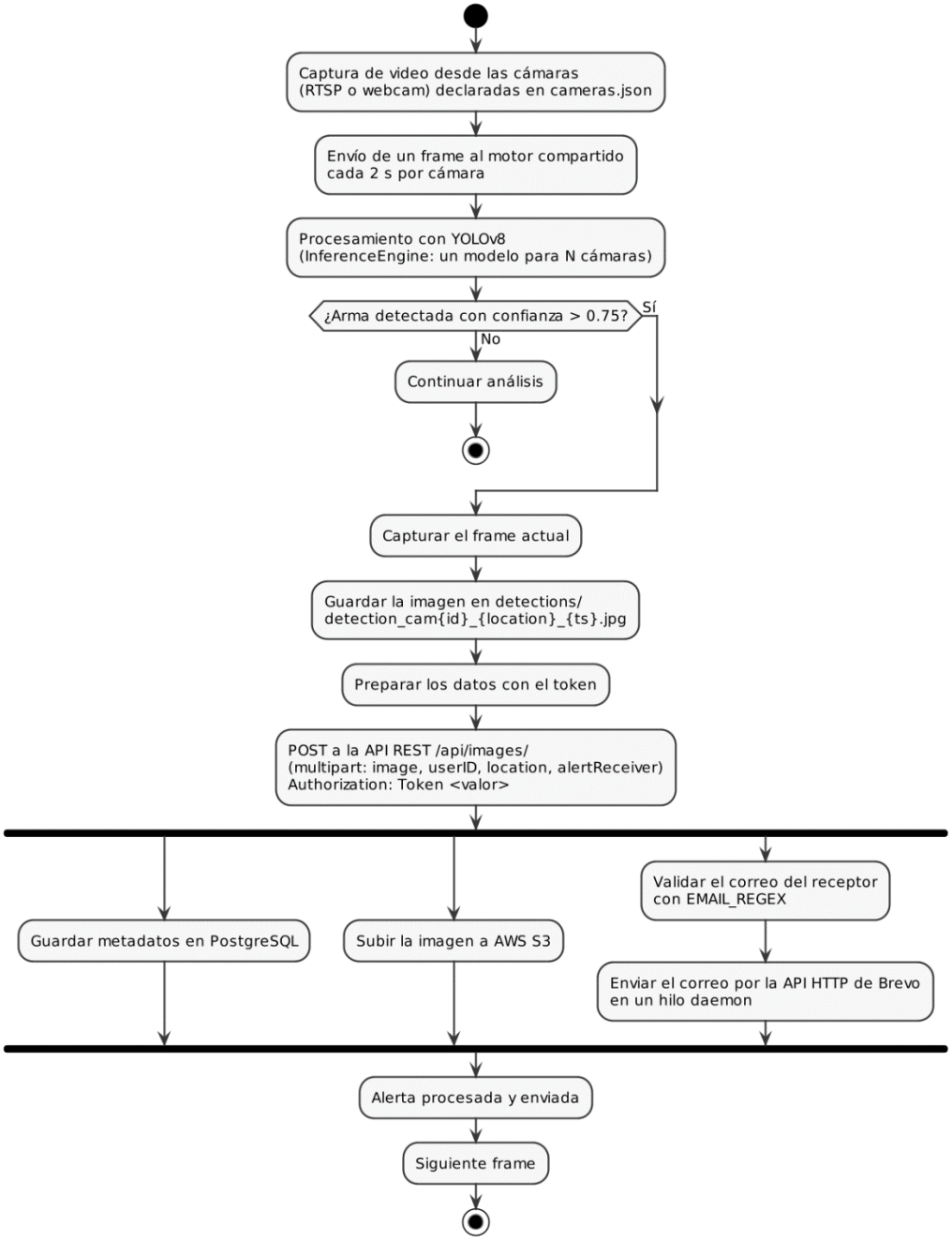
- Se envía un correo electrónico mediante la API HTTP de Brevo.

j. Visualización y seguimiento

El usuario puede acceder al dashboard web para consultar su historial de alertas, aplicar filtros y visualizar la evidencia correspondiente.

Figura 15:

Flujo de información del sistema de detección de armas



Nota. Esta arquitectura distribuida no solo garantiza la integridad, trazabilidad y disponibilidad de los datos, sino que permite una respuesta inmediata ante incidentes críticos.

2.7.2. Estructura del backend (API REST, Django ORM)

El backend del sistema de detección de armas está desarrollado con Django como framework principal y utiliza Django REST Framework (DRF) para construir una API RESTful robusta, segura y modular. La arquitectura se organiza bajo el patrón Model-View-Template (MVT) de Django, extendido con serializers para facilitar el manejo de datos estructurados y el intercambio eficiente con el cliente PyQt5.

a. Organización lógica del servidor

El servidor se encuentra segmentado en dos aplicaciones Django principales:

- **detection:** Administra la lógica de la interfaz web, autenticación de usuarios, visualización de alertas y filtros avanzados para la búsqueda de eventos.
- **alertuploadREST:** Encargada de la API REST que recibe alertas desde la aplicación de escritorio, valida la información recibida, persiste los datos y ejecuta el módulo de notificaciones (correo electrónico).

Esta separación permite aplicar principios de modularidad y separación de responsabilidades, facilitando el mantenimiento, pruebas unitarias y futuras escalabilidades del sistema.

b. Modelos Django

Los principales modelos implementados en el sistema son:

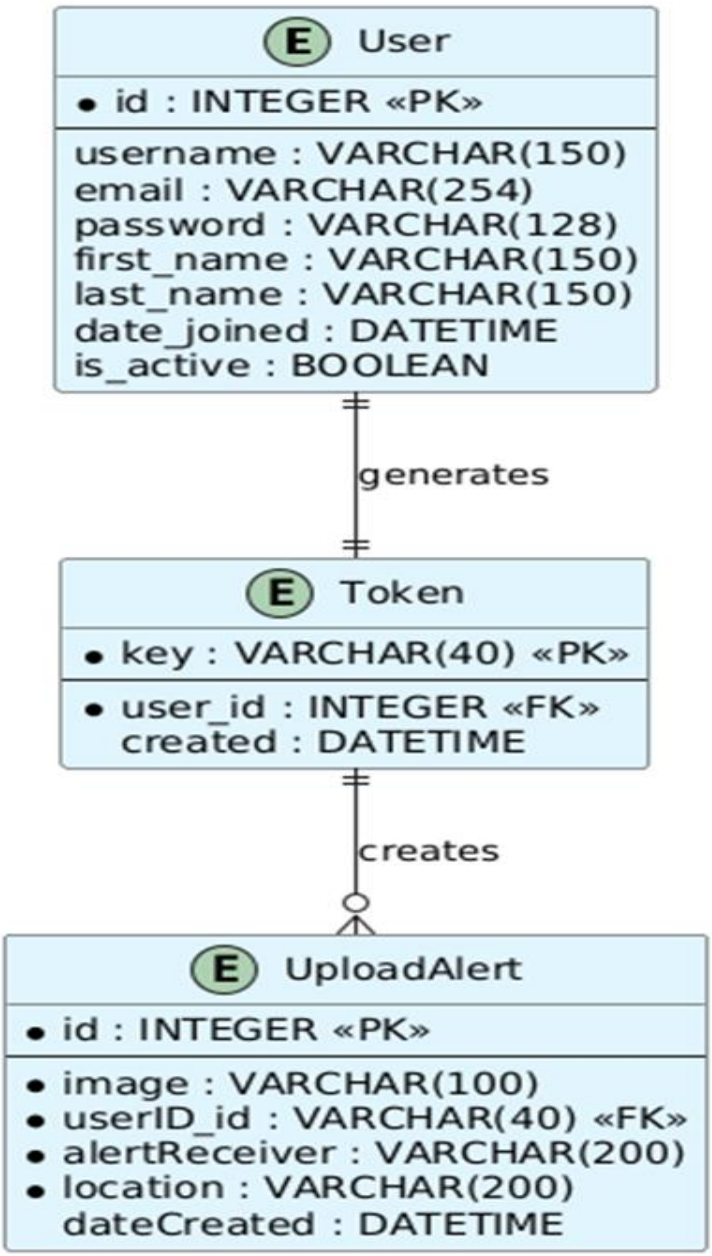
- **User:** Extiende el modelo estándar de Django con campos como username, email, password, first_name, last_name, date_joined, e is_active.
- **Token:** Modelo utilizado para la autenticación basada en tokens. Cada usuario tiene asociado un token único generado automáticamente al momento del registro.
- **UploadAlert:** Modelo central que registra los eventos de detección. Almacena la imagen capturada (image), el identificador del usuario (userID), la ubicación

(location), el receptor de la alerta (alertReceiver) y la fecha de creación (dateCreated).

La Figura 16 muestra cómo se relacionan los modelos mediante claves foráneas (user_id y userID_id) y cómo fluye la información desde la detección hasta su almacenamiento como alerta.

Figura 16:

Modelo entidad-relación (ERD) del sistema backend



Nota. Elaboración propia.

c. Endpoints de la API REST

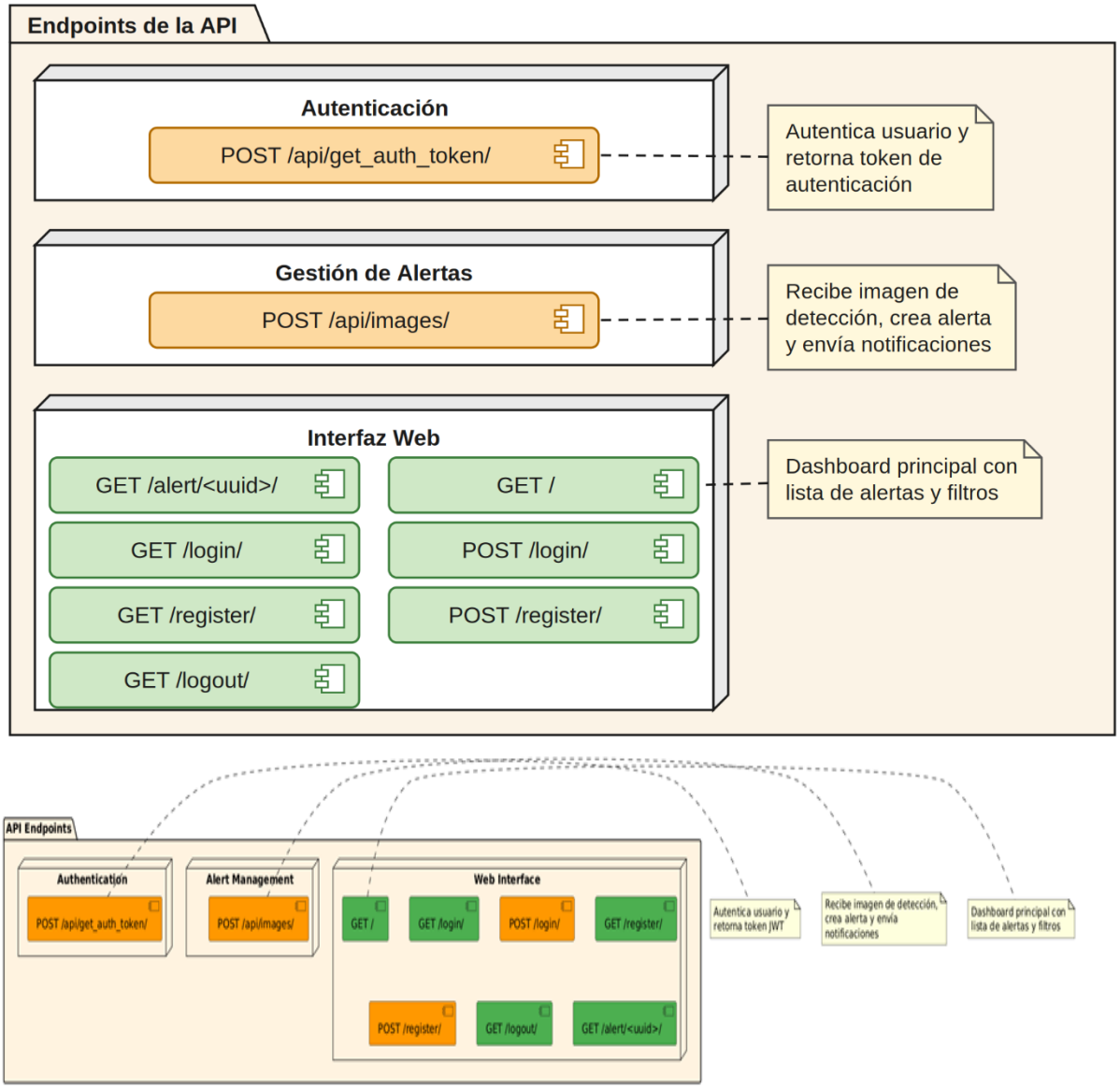
La API REST está diseñada para facilitar la comunicación entre el cliente PyQt5 y el servidor mediante solicitudes HTTP seguras. Los endpoints se agrupan según su funcionalidad principal:

- Autenticación:
    - **POST /api/get_auth_token/:** Recibe credenciales y retorna un token de autenticación de DRF (rest_framework.authtoken). Requiere que las siguientes peticiones se acompañen con el encabezado:
        - **Authorization:** Token <token_value>
- Gestión de alertas:
    - **POST /api/images/:** Recibe la alerta como multipart/form-data con el archivo image y los campos userID, location y alertReceiver. Realiza las validaciones, el almacenamiento y el envío automático de la notificación por correo electrónico.
- Interfaz Web:
    - GET /, GET /login/, GET /register/, GET /logout/, GET /alert/<uuid>/
    - POST /login/, POST /register/

Estos endpoints permiten tanto la autenticación de usuarios como la navegación por el dashboard y la gestión de alertas.

Figura 17:

Endpoints del servidor Django para autenticación, gestión de alertas y navegación web



Nota. Elaboración propia.

La Figura 17 presenta un esquema visual de los endpoints categorizados por dominio funcional, incluyendo los métodos HTTP correspondientes y su propósito dentro del sistema.

d. Manejo de autenticación

El sistema implementa un enfoque mixto de autenticación:

- **Autenticación en la API REST**: Basada en tokens, mediante Authorization: Token <token>. Los tokens se crean automáticamente al registrar un nuevo usuario usando señales post_save.
- **Autenticación en el dashboard web:** Utiliza sesiones de Django, permitiendo el manejo tradicional de login/logout con formularios y cookies de sesión seguras.

Este esquema dual permite separar claramente los contextos de uso: autenticación programática (cliente PyQt5) y navegación humana (dashboard web).

e. Base de datos

El sistema emplea PostgreSQL como motor de base de datos principal, desplegado en servicios cloud como Render. Su configuración incluye:

- Requisitos de conexión cifrada (SSL).
- Tiempo de espera optimizado.
- Conexiones persistentes con keepalive.

Las imágenes asociadas a las detecciones se almacenan de forma segura en Amazon S3, mientras que los metadatos se guardan en tablas relacionales, permitiendo consultas eficientes mediante ORM de Django y filtros avanzados.

2.7.3. Diseño del frontend: PyQt5 y web dashboard

El frontend está partido en dos piezas que atacan problemas distintos. La aplicación de escritorio en PyQt5 vive junto a las cámaras y ejecuta la detección en tiempo real; el dashboard en Django vive en el navegador y sirve para consultar, filtrar y gestionar las alertas después del evento. Ambas hablan con el mismo backend.

a. Aplicación de escritorio (PyQt5)

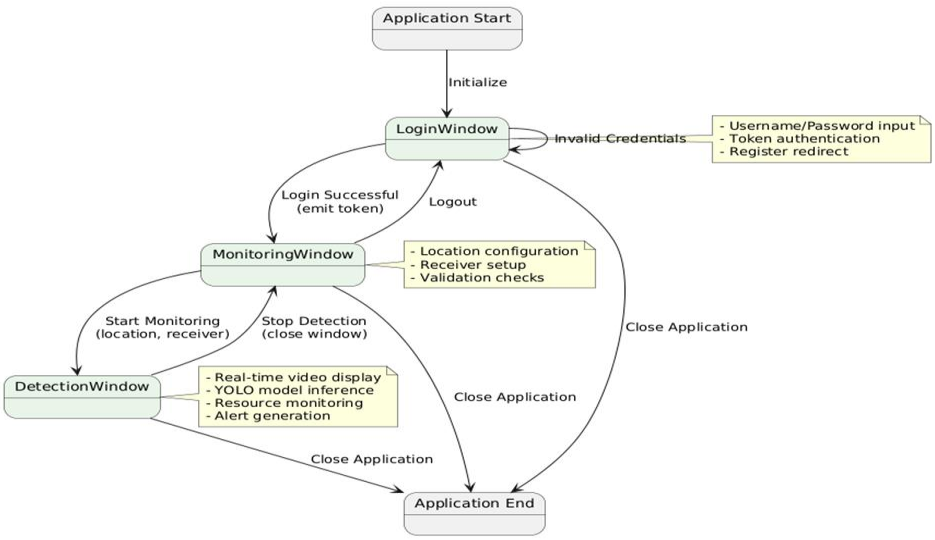
El cliente de escritorio está escrito en Python con PyQt5 y se organiza por ventanas, cada una con una responsabilidad concreta.

- **MainApplication** es el controlador que gestiona el ciclo de la aplicación y decide qué ventana se muestra en cada momento.
- **LoginWindow**. Si la autenticación pasa, el backend emite un token y la aplicación avanza a la siguiente etapa (véase el Manual de Usuario, Anexo E).
- **MonitoringWindow:** el usuario configura la operación: ubicación de monitoreo, datos de conexión de cada cámara y correo receptor de las alertas (véase el caso CP-CAM-011 del Anexo G).
- **DetectionWindow:** Muestra en una grilla el video en vivo de cada cámara habilitada con las bounding boxes superpuestas, lleva contadores de detección globales, monitorea CPU, RAM y GPU, y desde ella se inicia o detiene la inferencia con YOLOv8 (véase el caso CP-CAM-012 del Anexo G).

La Figura 18 resume la navegación entre ventanas: qué transiciones existen, qué eventos las disparan y qué acciones corren en cada etapa.

Figura 18:

Esquema de navegación - aplicación de escritorio (PyQt5).



Nota. La separación por ventanas deja espacio para módulos futuros —análisis forense, revisión offline— sin tocar el flujo de detección. Elaboración propia.

b. Dashboard web (Django)

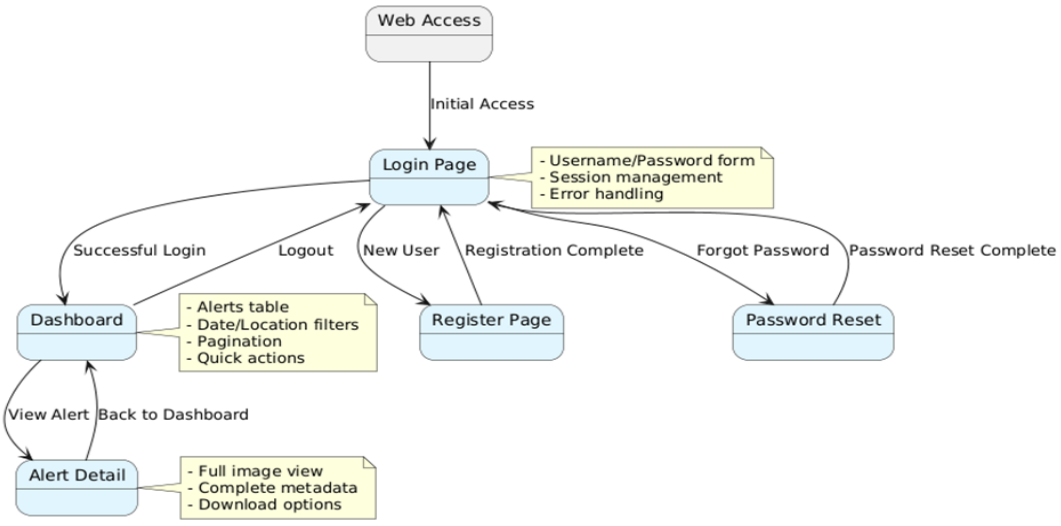
El dashboard web ofrece una interfaz intuitiva para la visualización, gestión y análisis de alertas.

- **Inicio de sesión y registro:** autenticación en todas las páginas, validación de formularios, mensajes de error y recuperación de contraseña por correo (Manual de Usuario, Anexo E).
- **Dashboard principal:** tabla de alertas paginada y ordenada por fecha, con miniatura de la detección, ubicación, receptor y timestamp (CP-DASH-002, Anexo G).
- **Filtros:** por rango de fechas, ubicación o destinatario, con django-filters (CP-DASH-003, Anexo G).
- **Vista de detalle:** imagen completa, metadatos y, si está habilitada, descarga del archivo (CP-DASH-004, Anexo G).

La Figura 19 muestra la navegación típica, del login al detalle de una alerta.

Figura 19:

Esquema de navegación del dashboard web del sistema



Nota. Este enfoque permite mantener la coherencia de experiencia entre usuarios técnicos y usuarios administrativos. Elaboración propia.

Capítulo III

3. Desarrollo e Implementación

3.1. Entorno de desarrollo y herramientas tecnológicas

El desarrollo del sistema de detección de armas se realizó utilizando un conjunto amplio y especializado de herramientas tecnológicas, seleccionadas en función de su robustez, compatibilidad y comunidad de soporte. Estas herramientas se clasificaron en función de su rol dentro de la arquitectura del sistema, abarcando desde el procesamiento de video en tiempo real en el cliente de escritorio hasta el backend web y el almacenamiento en la nube.

3.1.1. Lenguajes, frameworks y bibliotecas utilizadas

La Tabla 15 resume las tecnologías empleadas, detallando su categoría, versión implementada y propósito específico dentro del proyecto. Esta clasificación permite visualizar de forma estructurada los componentes clave que hicieron posible la construcción del sistema completo:

Tabla 15

Herramientas tecnológicas utilizadas en el desarrollo del sistema

Tecnología	Versión	Propósito
Lenguaje y cliente de escritorio		
Python	3.8.3	Lenguaje principal para toda la aplicación
PyQt5	5.15.11	Interfaz gráfica de la aplicación cliente
Visión por computadora e inteligencia artificial		
Ultralytics	8.2.60	Framework YOLOv8 para detección de objetos
PyTorch	2.3.1	Framework de deep learning
Torchvision	0.18.1	Utilidades de visión por computadora
OpenCV-Python / Headless	4.10.0.84	Procesamiento de video y cámara; versión headless en el servidor
Pillow	10.4.0	Procesamiento y manipulación de imágenes

Tecnología	Versión	Propósito
Análisis de datos y visualización		
NumPy	1.24.4	Operaciones matemáticas y arrays
SciPy	1.10.1	Algoritmos científicos y estadísticos
Pandas	2.0.3	Manipulación y análisis de datos
Matplotlib	3.7.5	Generación de gráficos y visualizaciones
Seaborn	0.13.2	Visualización estadística avanzada
Backend/Servidor		
Django	4.2.16	Framework web principal
Django REST Framework	-	API REST para comunicación cliente-servidor
django-filters	-	Filtrado avanzado en consultas
WhiteNoise	-	Servir archivos estáticos en producción
storages	-	Backends de almacenamiento (AWS S3)
PostgreSQL	-	Base de datos principal alojada en Render
psycopg (psycopg3)	3.2.3	Adaptador PostgreSQL para Python/Django
Comunicación y Redes		
Requests	2.32.3	Cliente HTTP para comunicación con APIs
psutil	6.0.0	Monitoreo de recursos del sistema (CPU, RAM)
GPUtil	1.4.0	Monitoreo de recursos de GPU
py-cpuinfo	9.0.0	Información detallada del procesador
Optimización y Rendimiento		
Brevo (Sendinblue) API	-	Envío de correos electrónicos transaccionales por HTTP
Frontend Web		
HTML5, CSS3 y JavaScript	-	Estructura, estilos e interactividad del dashboard
Bootstrap	4.3.1	Framework CSS responsivo
Font Awesome	6.0.0	Biblioteca de iconos
Infraestructura y entrenamiento		

Tecnología	Versión	Propósito
Render	**-**	**Plataforma de hosting web y base de datos**
Google Colab	-	Entrenamiento del modelo en la nube
Google Drive	-	Almacenamiento de datasets y modelos
AWS S3	-	Almacenamiento de imágenes en la nube

Nota. Elaboración propia.

Como se evidencia en la Tabla 14, el sistema integra múltiples niveles tecnológicos: en el cliente se utiliza PyQt5 para la construcción de interfaces gráficas de escritorio, junto con bibliotecas como OpenCV y Ultralytics YOLOv8 para el procesamiento visual e inferencia de modelos de IA. En el backend, Django y Django REST Framework se emplean para la creación de servicios web y APIs RESTful seguras, mientras que PostgreSQL y AWS S3 se encargan del almacenamiento estructurado y no estructurado de datos.

Además, se incorporan librerías de soporte para análisis numérico, visualización de datos, monitoreo del sistema, comunicaciones seguras, y optimización del rendimiento, lo que garantiza una solución integral, eficiente y escalable.

3.1.2. Justificación técnica de la pila de desarrollo

El autor eligió cada tecnología contra cinco criterios: que corriera en más de un sistema operativo; que resolviera la inferencia con eficiencia; que tuviera documentación y comunidad activas; que él mismo ya la hubiera usado; y que cubriera los requisitos funcionales y no funcionales fijados en la sección anterior.

Las categorías siguientes justifican, una por una, la elección de cada componente:

a. Lenguaje Principal: Python 3.x

- **Ecosistema de IA/ML:** PyTorch, OpenCV y Ultralytics —las tres bibliotecas centrales del sistema— exponen Python como interfaz principal; ninguna alternativa en otro lenguaje cubre las tres.
- **Compatibilidad: el** mismo código corre en Windows, Linux y macOS. Importa aquí porque los equipos de los dos locales no comparten sistema operativo.
- **Integración vertical: un solo lenguaje resuelve la captura** de video, la inferencia, el backend web y el envío de correo, sin puentes entre entornos que sumen latencia o puntos de falla.
- **Mantenimiento: la** sintaxis abarata retomar el código meses más tarde, algo que un trabajo de titulación repartido en fases exige de verdad.

b. Framework de IA: YOLOv8 (Ultralytics)

- **Requisito de partida: detectar** un arma en tiempo real exige una latencia de inferencia menor a 100 ms por frame. De las arquitecturas revisadas en el Capítulo I, solo la familia YOLO cumple esa cota en hardware de gama media.
- **Transfer learning: el entrenamiento partió de pesos preentrenados y ajustó el** modelo por fine-tuning. Así bastó un dataset propio de tamaño acotado para llegar a 90,3 % de precisión, 89,4 % de recall y 91,2 % de mAP@0.5.
- **API:** la interfaz de Ultralytics resuelve entrenamiento, validación e inferencia con las mismas llamadas, lo que acortó la integración con el cliente PyQt5.
- **Hardware:** soporte nativo de CUDA para GPU y modo CPU optimizado. El sistema aprovecha lo que haya en el equipo del local sin cambiar código.

c. Interfaz de Usuario: PyQt5

- **Rendimiento nativo**: Aplicación desktop con acceso directo a recursos del sistema sin overhead de navegador.

- **Capacidades multimedia**: Soporte robusto para streaming de video y procesamiento en tiempo real.
- **Compatibilidad multiplataforma**: Una sola base de código para Windows, Linux y macOS.
- **Madurez del framework**: PyQt5 es estable, bien documentado y ampliamente utilizado en aplicaciones profesionales.
- **Requisitos del sistema**: Necesidad de monitoreo de recursos del sistema (CPU, GPU, RAM) y manejo eficiente de video.

d. Backend Web: Django + Django REST Framework

- **Desarrollo rápido**: "Batteries included" - ORM (Object-Relational Mapping), autenticación, admin panel incluidos.
- **Seguridad**: Protecciones integradas contra falsificación de petición en sitios cruzados (CSRF), inyección SQL y ejecución de scripts entre sitios (XSS).
- **Escalabilidad**: Arquitectura MVT que facilita el crecimiento del sistema.
- **API REST robusta**: Django REST Framework proporciona serialización, autenticación por tokens y documentación automática.
- **Experiencia previa**: Framework familiar que reduce el tiempo de desarrollo.

e. Base de Datos: PostgreSQL en Render.

- **Robustez empresarial**: PostgreSQL maneja transacciones ACID y concurrencia alta.
- **Escalabilidad**: Soporte para millones de registros sin degradación de performance.
- **Tipos de datos avanzados**: JSON (JavaScript Object Notation), arrays, geometría - útiles para metadatos de detecciones.
- **Compatibilidad con Django**: Excelente integración con el ORM de Django.

- **Hosting en Render**:Configuración automática y mantenimiento, backups automáticos, SSL/TLS integrado, escalabilidad automática

f. Procesamiento de Imágenes: OpenCV + Pillow

- **OpenCV**: Estándar de la industria para computer vision, optimizado para tiempo real.
- **Eficiencia**: Implementaciones en C/C++ con bindings Python para máximo rendimiento.
- **Funcionalidades completas**: Captura de cámara, procesamiento de frames, codecs de video.
- **Compatibilidad**: Soporte para múltiples formatos de imagen y backends de cámara.
- **Pillow**: Complementa OpenCV para manipulación de imágenes estáticas y conversiones de formato.

g. Cálculo Científico: NumPy + SciPy + Pandas

- **Rendimiento**: Operaciones vectorizadas en C que superan Python puro por 10-100x.
- **Ecosistema PyTorch**: NumPy es la base de PyTorch, garantizando compatibilidad total.
- **Análisis de datos**: Pandas facilita el procesamiento de logs y métricas del sistema.
- **Optimizaciones Intel**: MKL, OpenMP y TBB proporcionan paralelización automática.

h. Visualización: Matplotlib + Seaborn

- **Flexibilidad**: Matplotlib permite gráficos personalizados para métricas del sistema.
- **Calidad profesional**: Gráficos publication-ready para reportes y análisis.

- **Seaborn**: Simplifica visualizaciones estadísticas complejas.
- **Integración**: Funciona seamlessly con Pandas y NumPy.

i. Servicio de Notificación: API HTTP de Brevo

- Envío sobre HTTPS:
  - Puerto 443, camino que Render no bloquea, a diferencia del Simple Mail Transfer Protocol (SMTP) saliente por el 587.
  - Correo HTML con respaldo en texto plano y envío asíncrono.
  - Panel de entregabilidad y verificación del remitente.
- Operación y costo:
  - Plan gratuito suficiente para el volumen de alertas del piloto.
  - Credenciales fuera del código, en variables de entorno.
  - No requiere mantener un cliente SMTP en el servidor.

j. Almacenamiento: AWS S3

- **Escalabilidad**: Almacenamiento ilimitado para imágenes de detecciones.
- **Durabilidad**: 99,999999999 % (11 9's) de durabilidad de datos.
- **CDN integrado**: CloudFront para entrega rápida de imágenes globalmente.
- **Costo-efectivo**: Pago por uso, económico para almacenamiento de archivos multimedia.
- **Integración Django**: Storages backends facilitan la integración.

k. Plataforma de Entrenamiento: Google Colab.

- **GPU gratuita**: Acceso a Tesla T4/V100 sin costo.
- **Entorno preconfigurado**: PyTorch, CUDA y bibliotecas ML ya instaladas.
- **Colaboración**: Notebooks compartibles para trabajo en equipo.

- **Integración Google Drive**: Almacenamiento directo de datasets y modelos.
- **Sin configuración**: Elimina complejidad de setup de entorno ML.

l. Hosting: Render.

- **Simplicidad**: Deploy directo desde Git sin configuración compleja.
- **Escalabilidad automática**: Ajuste de recursos según demanda.
- **SSL/HTTPS integrado**: Seguridad sin configuración adicional.
- **Base de datos incluida**: PostgreSQL managed sin administración.
- **Costo-efectivo**: Tier gratuito para desarrollo y planes accesibles para producción.

m. Criterios de decisión consolidados

A partir de la evaluación de las herramientas seleccionadas, se destacan los siguientes criterios que guiaron la construcción de la pila tecnológica del sistema (Tabla 16).

Tabla 16:

Criterios de decisión.

Criterio	Justificación
Compatibilidad	Soporte multiplataforma, APIs abiertas y estándares industriales.
Eficiencia	Soporte para aceleración por GPU, paralelización y bajo consumo de recursos.
Disponibilidad	Amplia documentación, comunidades activas, y abundantes ejemplos.
Experiencia previa	Uso de frameworks familiares que reducen la curva de aprendizaje.
Requisitos técnicos	Soporte para tiempo real, detección de armas, notificaciones automáticas, UI responsiva.

Nota. Elaboración propia.

n. Arquitectura resultante.

Gracias a esta selección tecnológica estructurada, el sistema ofrece:

- Detección en tiempo real con latencia mínima (<100 ms por frame)
- Notificaciones automáticas y procesamiento distribuido
- Escalabilidad horizontal para múltiples usuarios y cámaras
- Alta disponibilidad y seguridad en producción

3.2. Implementación del modelo de IA

3.2.1. Selección y entrenamiento de YOLOv8

El modelo de detección seleccionado para el sistema fue YOLOv8, desarrollado por Ultralytics, una de las versiones más recientes y optimizadas de la familia You Only Look Once. Esta elección responde a una combinación de criterios clave: precisión en la detección de objetos, velocidad de inferencia en tiempo real y compatibilidad con entornos de implementación modernos como PyTorch, Google Colab y sistemas con GPU.

a. Justificación técnica de la elección del modelo

La elección de YOLOv8 se decidió contra cuatro candidatos: YOLOv5, YOLOv7, Faster R-CNN y SSD. Pesaron estos factores:

- **Inferencia en tiempo real:** La arquitectura optimizada de YOLOv8 permite procesar video en vivo con baja latencia, siendo ideal para aplicaciones de videovigilancia activa.
- **Eficiencia computacional:** Reduce el tamaño del modelo sin sacrificar precisión, permitiendo su ejecución en entornos de hardware limitado.
- **Ajuste fino (fine-tuning):** Incluye mecanismos flexibles para configurar hiperparámetros según la naturaleza del dataset.

- **Integración simplificada:** La implementación a través de la librería Ultralytics facilita el entrenamiento, validación cruzada y el monitoreo de métricas desde un entorno Python.

b. Dataset utilizado

Para entrenar el modelo, se construyó un dataset personalizado a partir de fuentes públicas y materiales generados específicamente para el proyecto. Este conjunto de datos contiene imágenes que simulan contextos reales de vigilancia donde pueden aparecer armas como pistolas y cuchillos.

- **Tamaño total:** 4.000 imágenes etiquetadas.
- **Clases: handGun** (2.050 imágenes) y **knife** (1.950 imágenes).
- **División del dataset:** Entrenamiento: 60% (2.400 imágenes); **Validación:** 20% (800 imágenes); **Prueba:** 20% (800 imágenes)

c. Estructura de anotaciones y archivo de configuración

Las anotaciones estan en el formato estándar de YOLO, donde cada imagen está asociada a un archivo .txt con anotaciones normalizadas en el formato:

<class_id> <x_center> <y_center> <width> <height>

El archivo de configuración general data.yaml definió las rutas relativas y las clases del dataset:

Figura 20:

Configuración archivo data.yaml

```
path: /content/gdrive/MyDrive/yolov8-optimized/data/datasetevaluate

train: images/train
val: images/val
test: images/test

names: ['handGun','knife']
```

Nota. Elaboración propia.

d. Entorno de entrenamiento

El entrenamiento se llevó a cabo en Google Colab, aprovechando la aceleración por GPU (Tesla T4) y un entorno preconfigurado con las bibliotecas necesarias:

- **Framework:** Ultralytics YOLOv8 v8.3.4
- **Python:** 3.10.12
- RAM disponible: 12.7 GB
- **GPU:** Tesla T4 (15.102 MiB)
- Duración del entrenamiento: 14 horas

e. Configuración de hiperparámetros

La Tabla 17 resume los valores usados en el entrenamiento del modelo YOLOv8:

Tabla 17:

Hiperparámetros de entrenamiento

Parámetro	Valor	Descripción
epochs	800	Número total de épocas
batch	16	Tamaño de lote
imgsz	640	Resolución de entrada de imágenes
lr0	0.01	Tasa de aprendizaje inicial
lrf	0.01	Tasa de aprendizaje final
momentum	0.95	Momento del optimizador
weight_decay	0.0001	Regularización L2
patience	75	Early stopping
optimizer	SGD	Optimizador (Gradiente descendente)
workers	8	Número de hilos para carga de datos

Nota. Elaboración propia.

f. Aumento de datos (Data Augmentation)

Para mejorar la generalización del modelo, se aplicaron técnicas de aumento durante el entrenamiento, resumidas en la Tabla 18:

Tabla 18:

Técnicas de aumento de datos

Técnica	Parámetro	Valor	Propósito
HSV Hue	hsv_h	0.015	Variación de tonalidad
HSV Saturation	hsv_s	0.8	Modificación de saturación
HSV Value	hsv_v	0.1	Cambio de brillo
Translación	translate	0.1	Desplazamiento de objetos
Escalado	scale	0.7	Variación de tamaño
MixUp	mixup	1.0	Mezcla de imágenes
Flip horizontal	fliplr	0.5	Inversión horizontal
Shear	shear	25	Transformación tipo “corte”
Perspectiva	perspective	0.001	Distorsión espacial leve

Nota. Elaboración propia.

Esta configuración permitió obtener un modelo robusto, entrenado para identificar armas en condiciones diversas, preparado para integrarse directamente con el sistema cliente y servidor descrito en capítulos anteriores.

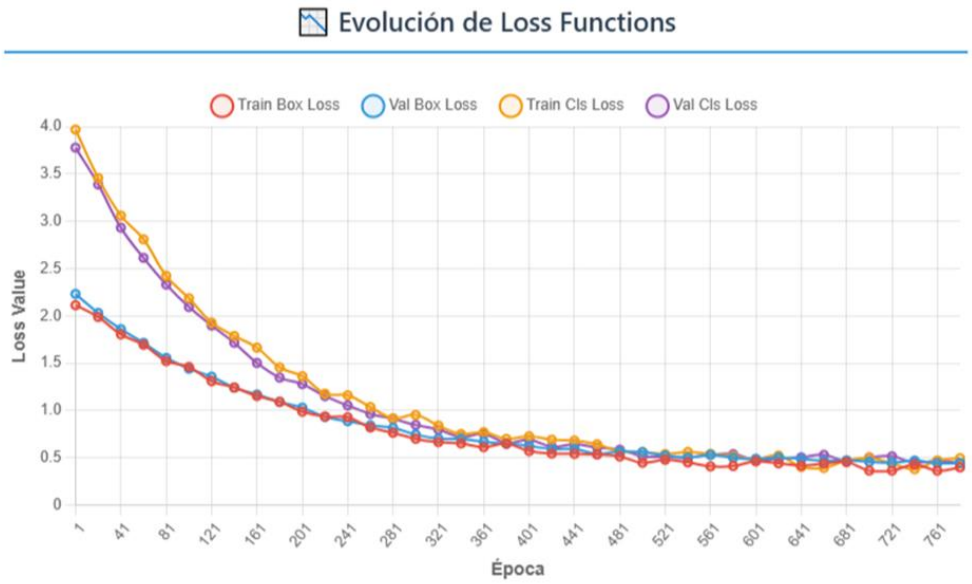
3.2.2. Ajuste fino (fine-tuning) y validación del modelo

El proceso de ajuste fino del modelo de detección se realizó a partir de una estrategia de transfer learning utilizando los pesos preentrenados de YOLOv8 sobre el dataset COCO. Esta técnica permitió mejorar la especialización del modelo en la detección de armas (pistolas y cuchillos) manteniendo una base sólida de conocimientos generales previamente aprendidos.

Durante el proceso de fine-tuning, se aplicaron prácticas recomendadas como congelamiento inicial de capas, descongelamiento progresivo y optimización personalizada de hiperparámetros. El modelo fue entrenado durante 800 épocas con técnicas avanzadas de aumento de datos (como mixup, shear, flip y transformaciones HSV), logrando una convergencia rápida y estable, como se observa en la Figura 21.

Figura 21:

Evolución de Loss Functions

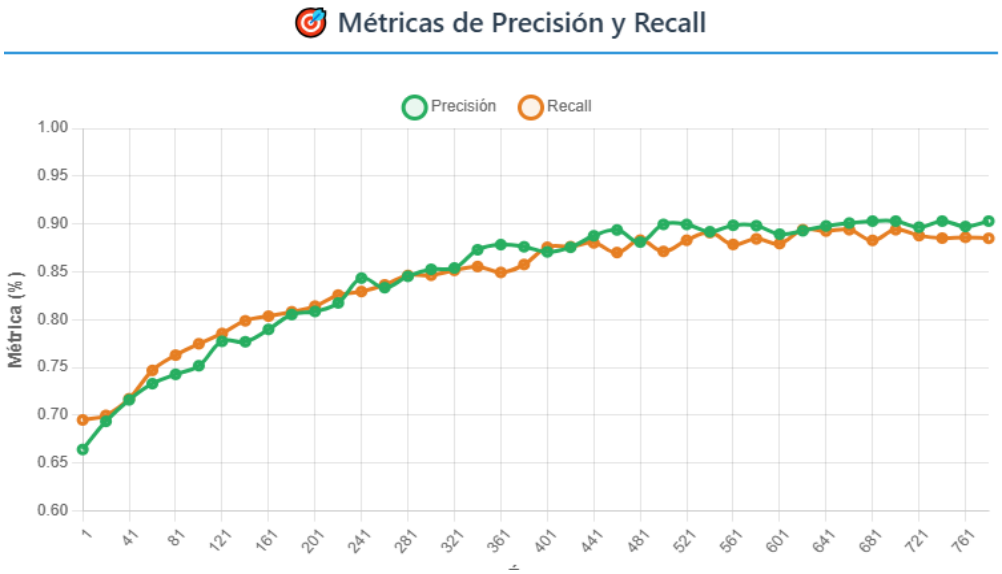


Nota. Elaboración propia.

La Figura 22 muestra la evolución de precisión y recall: ambas superan el 89 % y se estabilizan sin señales de sobreajuste. La Figura 23 recoge el mAP, con 91,2 % en mAP@0.5 y 68,4 % en mAP@0.5:0.95. Ese segundo valor es el exigente, porque castiga las cajas predichas que se solapan mal con la real.

Figura 22:

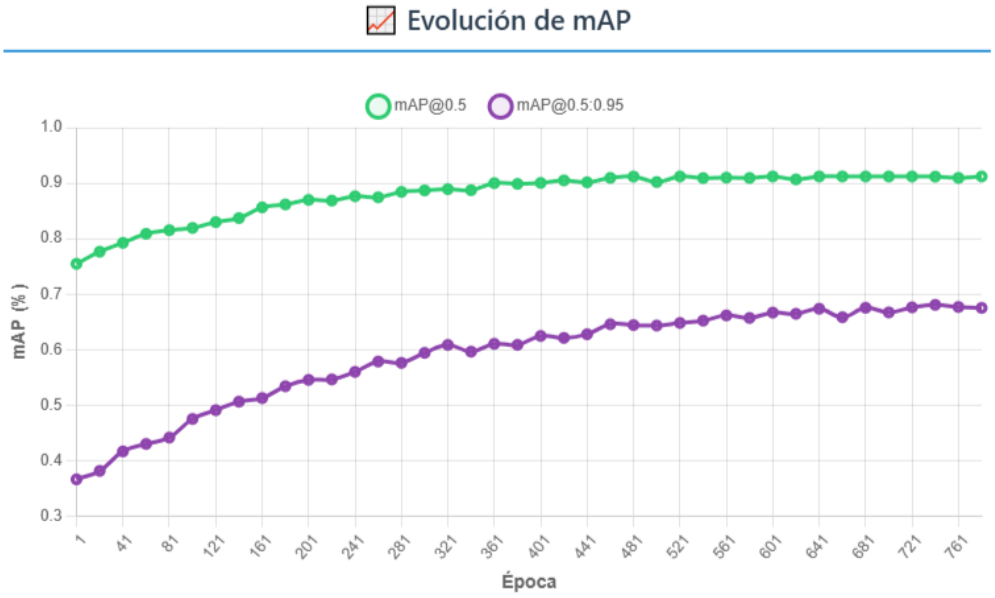
Métricas de Precisión y Recall



Nota. Elaboración propia.

Figura 23

Evolución de mAP

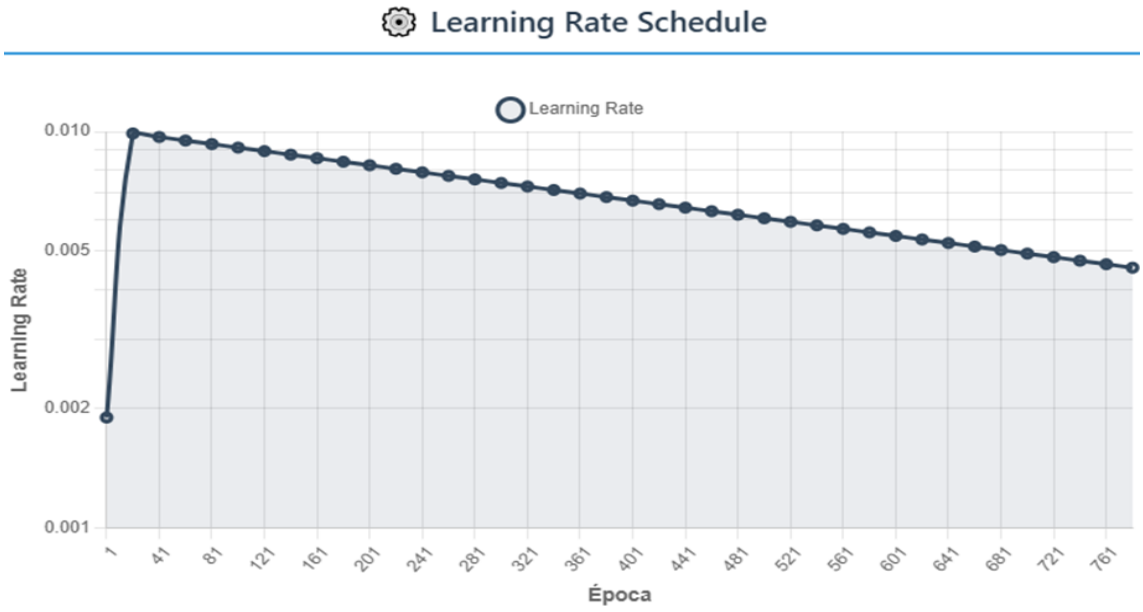


Nota. Elaboración propia.

La Figura 24 grafica el learning rate. El entrenamiento lo redujo de forma progresiva según una programación descendente, y el aprendizaje se estabilizó hacia las últimas épocas.

Figura 24:

Learning Rate Schedule



Nota. Elaboración propia.

La Tabla 19 resume las métricas que dejó el ajuste fino:

Tabla 19:

Métricas finales del modelo YOLOv8 tras fine-tuning

Métrica	Valor Final	Mejora vs. Baseline	Interpretación
Precision	90,3 %	+12,1 % (78,2 %)	90,3 % de detecciones fueron correctas
Recall	89,4 %	+9,6 % (79,8 %)	89,4 % de objetos reales fueron detectados
mAP@0.5	91,2 %	+5,9 % (85,3 %)	Precisión media con IoU ≥ 0.5
mAP@0.5:0.95	68,4 %	+25,7 % (42,7 %)	Precisión media en todo el rango de IoU
Val Box Loss	0.403	-77,5 % (1.790)	Error en la predicción de bounding boxes
Val Cls Loss	0.423	-87,4 % (3.361)	Error en clasificación de clases

Nota. Elaboración propia.

Las métricas confirman que el modelo generaliza sin caer en sobreajuste. Además, se validaron los resultados utilizando validación cruzada estratificada para asegurar una evaluación robusta sobre diferentes subconjuntos del dataset.

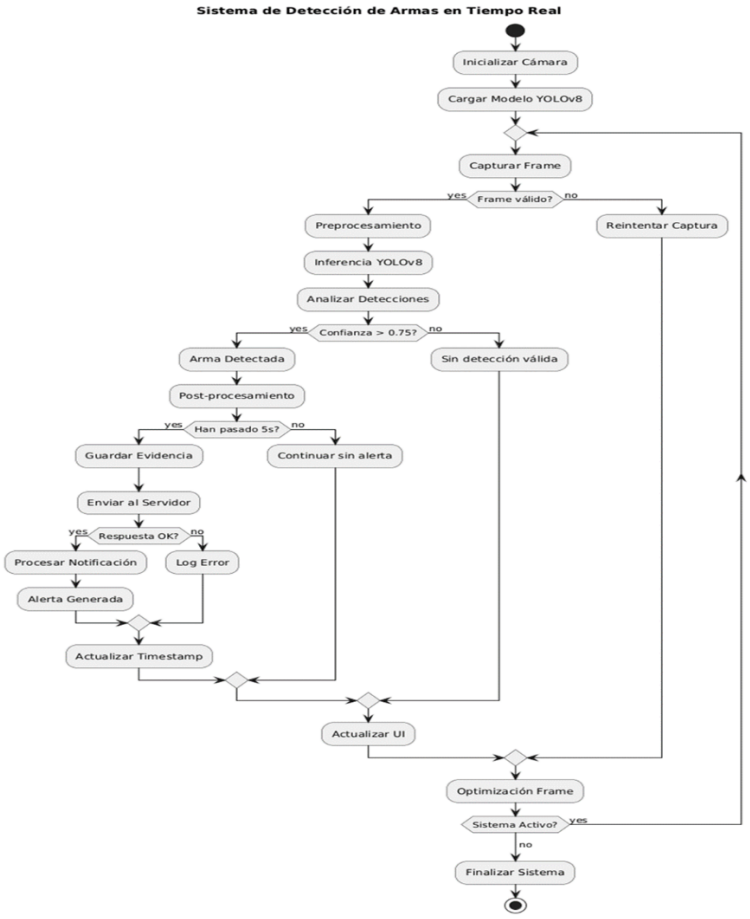
3.2.3. Integración con flujo de captura en tiempo real

La integración del modelo de IA con la cámara y el cliente se realiza mediante un flujo optimizado que permite la detección de armas en tiempo real. Este proceso une la captura de video desde hardware físico, el análisis por parte del modelo YOLOv8 y la generación de alertas hacia el servidor backend, todo bajo condiciones de latencia mínima.

a. Arquitectura del flujo de procesamiento

Figura 25:

Diagrama de flujo en tiempo real



Nota. Elaboración propia.

El Diagrama de flujo del proceso en tiempo real (Figura 25) resume esta lógica, desde la inicialización de la cámara hasta la validación del resultado, el guardado de evidencia, y el envío al servidor para procesar la alerta.

b. Proceso de conexión cámara-modelo-cliente

Este flujo se reparte entre dos componentes. DetectionTapo levanta un hilo por cámara y encapsula la captura y el envío de evidencia; InferenceEngine centraliza el modelo YOLOv8 y atiende por turnos a todas las cámaras registradas:

1. **Inicialización del sistema:** Se definen parámetros como la ubicación del modelo, token, receptor de alerta y cámara (véase el apartado 6.1 del Manual del Programador, Anexo F).
2. **Captura y procesamiento continuo:** Dentro de un hilo (run()), se captura cada frame, se aplica inferencia con el modelo y se llama al procesamiento condicional (véanse los apartados 7.1 y 8.1 del Manual del Programador, Anexo F).
3. **Procesamiento de detecciones:** Se evalúa la confianza (conf) y, si supera el umbral de 0,75, se genera una alerta (véase el apartado 7.2 del Manual del Programador, Anexo F).

c. Componentes técnicos de la integración

- Captura de video:

Utiliza OpenCV con soporte para múltiples backends: DirectShow o MSMF para las cámaras web USB y FFmpeg para los flujos RTSP de las cámaras IP (Tapo, Hikvision, Dahua, Reolink o URL manual), con buffer mínimo para reducir la latencia. La resolución es configurable, aunque se optimiza para 640x640 px, y el sistema ajusta automáticamente los FPS al rendimiento del hardware.

- Procesamiento del modelo:
    - **Entrada:** imagen RGB normalizada

- **Salida:** bounding boxes, clases, confianza
- Umbral configurable: 0.75

- Optimizaciones implementadas:
  - **Throttling temporal:** todos los frames se muestran en la interfaz, pero solo se envía uno al motor de inferencia cada 2 segundos por cámara
  - Intervalo de captura y envío: mínimo 4 segundos
  - **Gestión de memoria:** liberación automática tras cada batch
  - **Recuperación de errores:** reinicio de cámara ante fallos

d. Comunicación cliente-servidor

Cuando se detecta una amenaza, se genera una alerta que se transmite mediante una solicitud HTTP segura al servidor Django.

- Datos enviados

Figura 26:

Estructura de datos enviados

```
data = {
    'userID': self.token,
    'location': self.location,
    'alertReceiver': self.receiver,
    'image': frame_captured
}
```

Nota. Elaboración propia.

- Protocolo de comunicación:
  - Método: POST
  - **Endpoint:** /api/images/
  - **Autenticación:** cabecera Authorization Token <valor>

- **Formato:** multipart/form-data (imagen y metadatos)
- **Timeout:** 10 s por intento

e. Monitoreo de rendimiento

El sistema incluye un módulo de monitoreo en tiempo real para asegurar estabilidad:

- Uso de CPU y RAM
- Utilización de GPU (si disponible)
- FPS alcanzado por el hilo de detección
- Latencia promedio de red

Este monitoreo permite ajustar dinámicamente la velocidad de análisis y el intervalo de captura en función de los recursos disponibles.

3.3. Desarrollo de la interfaz cliente (PyQt5)

3.3.1. Visualización de video y detecciones

a. Arquitectura de visualización

La interfaz de usuario del sistema fue desarrollada utilizando PyQt5, implementando un módulo de visualización en tiempo real que integra la captura de video de una o varias cámaras —IP sobre RTSP o webcam USB— con el proceso de inferencia del modelo de detección YOLOv8. Esta funcionalidad está encapsulada dentro de un flujo asincrónico controlado por hilos (threads), con un hilo DetectionTapo por cámara, para garantizar la fluidez y respuesta de la interfaz gráfica de usuario (GUI).

b. Componente principal de visualización

El elemento visual clave es un objeto QLabel llamado CameraLabel, que actúa como contenedor para el stream de video procesado. Este componente se replica una vez por cámara dentro de la clase DetectionWindowDual (archivo detectionWindowDual.py), que los organiza en una grilla adaptable, y es configurado con

estilos personalizados para mantener una presentación profesional y centrada dentro de la ventana principal.

Figura 27:

Inicialización de CameraLabel en detectionWindowDual.py

```
self.CameraLabel = QLabel(central_widget)
self.CameraLabel.setAlignment(Qt.AlignCenter)
self.CameraLabel.setStyleSheet(DetectionWindowStyles.CAMERA_LABEL)
```

Nota. Elaboración propia.

c. Procesamiento y visualización de detecciones

El sistema sigue una arquitectura estructurada que comprende los siguientes pasos:

Captura de frames, inferencia del modelo, superposición visual y Renderizado

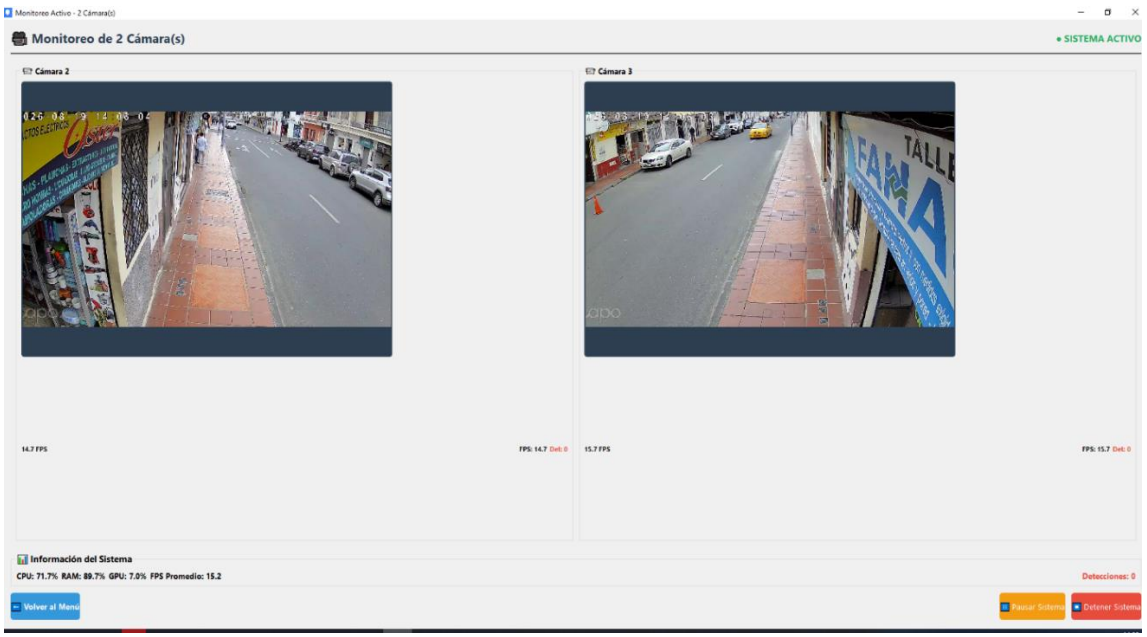
Para analizar el fragmento del método, véase el apartado 7.2 del Manual del Programador, Anexo F.

d. Visualización en la interfaz de usuario

La interfaz muestra el video en una ventana de monitoreo denominada Weapon Detection System (Figura 28). El estado del sistema (activo/inactivo) se indica con un selector, acompañado de un contador de detecciones global y otro por cámara, y en el lado derecho se incluyen paneles que reportan el uso en tiempo real de CPU, memoria RAM y GPU.

Figura 28:

Interfaz de monitoreo mostrando el stream en vivo y recursos del sistema



Nota. Elaboración propia.

e. Conversión y presentación del frame

Para mostrar el frame dentro del QLabel, se convierte el resultado procesado en un objeto QPixmap mediante la función setImage, utilizando una transformación suave y manteniendo la relación de aspecto (véase el apartado 8.6 del Manual del Programador, Anexo F)

f. Optimización de rendimiento y recursos

Con el fin de mantener un rendimiento estable durante la operación continua, se aplicaron las siguientes optimizaciones:

- **Throttling temporal:** el análisis se limita a un frame cada 2 segundos por cámara para evitar la sobrecarga, mientras la visualización conserva todos los frames.
- **Control thread-safe:** Uso de banderas y eventos para sincronización segura de recursos compartidos.

- **Liberación de memoria:** Eliminación automática de recursos al finalizar la detección o cerrar la aplicación.

Este enfoque asegura una visualización eficiente y estable, adecuada para entornos de vigilancia en tiempo real donde la inmediatez de la respuesta es crítica.

3.3.2. Controles y envío de alertas

a. Arquitectura de controles

La interfaz cliente desarrollada en PyQt5 se estructura en torno a tres ventanas principales, cada una con funciones claramente definidas dentro del ciclo de vida de detección:

- **LoginWindow:** Permite la autenticación segura del usuario mediante validación de credenciales contra la API REST.
- **MonitoringWindow:** Actúa como interfaz de configuración inicial, donde el usuario define parámetros clave como la ubicación, los datos de conexión de cada cámara y el correo electrónico receptor de alertas.
- **DetectionWindow:** Controla la visualización en tiempo real, así como la ejecución y terminación del proceso de monitoreo activo.

b. Controles de configuración inicial

En la ventana MonitoringWindow, el usuario tiene acceso a controles de interfaz que permiten preparar el sistema antes de iniciar el monitoreo. Estos controles están sujetos a validaciones específicas que aseguran la correcta inicialización del flujo (Tabla 20).

Tabla 20:

Detalle acerca de controles del sistema de monitoreo

Control	Tipo	Función	Validación
Campo Ubicación	QLineEdit	Define la ubicación de monitoreo	Mínimo 3 caracteres
Campo Contacto	QLineEdit	Dirección de correo electrónico del receptor	Validación de formato de correo electrónico
Botón Iniciar Monitoreo	QPushButton	Inicia el proceso de detección	Requiere campos válidos
Botón Cerrar Sesión	QPushButton	Finaliza sesión y redirige al login	Confirmación si monitoreo activo
Botón Stop Monitoring	QPushButton	Detiene el monitoreo en curso	Disponible solo en detección activa
Botón Restart Monitoring	QPushButton	Reinicia el monitoreo tras detenerlo	Aparece post-detección
Botón Back	QPushButton	Regresa a ventana anterior	Solo habilitado al detener

Nota. Elaboración propia.

c. Sistema de validación en tiempo real

La aplicación incluye validación dinámica de los campos de entrada mediante expresiones regulares y manejo de estados de botones. Esto garantiza que el sistema solo se pueda activar cuando los parámetros son correctos, reduciendo errores en la operación (Figura 29).

d. Mecanismo de envío de alertas

El sistema genera y envía alertas de forma automática cuando se detecta un arma con una confianza superior al umbral configurado (0,75). Este proceso no interrumpe la detección ni bloquea la interfaz, gracias al uso de hilos secundarios para el envío HTTP.

e. Flujo de envío al servidor (API Django)

- Captura del frame con arma detectada
- Ensamblaje de datos: imagen + metadatos (token, ubicación, receptor)
- Envío vía POST a /api/images/ con la cabecera Authorization: Token
- Procesamiento en el backend para validar, registrar y notificar la alerta

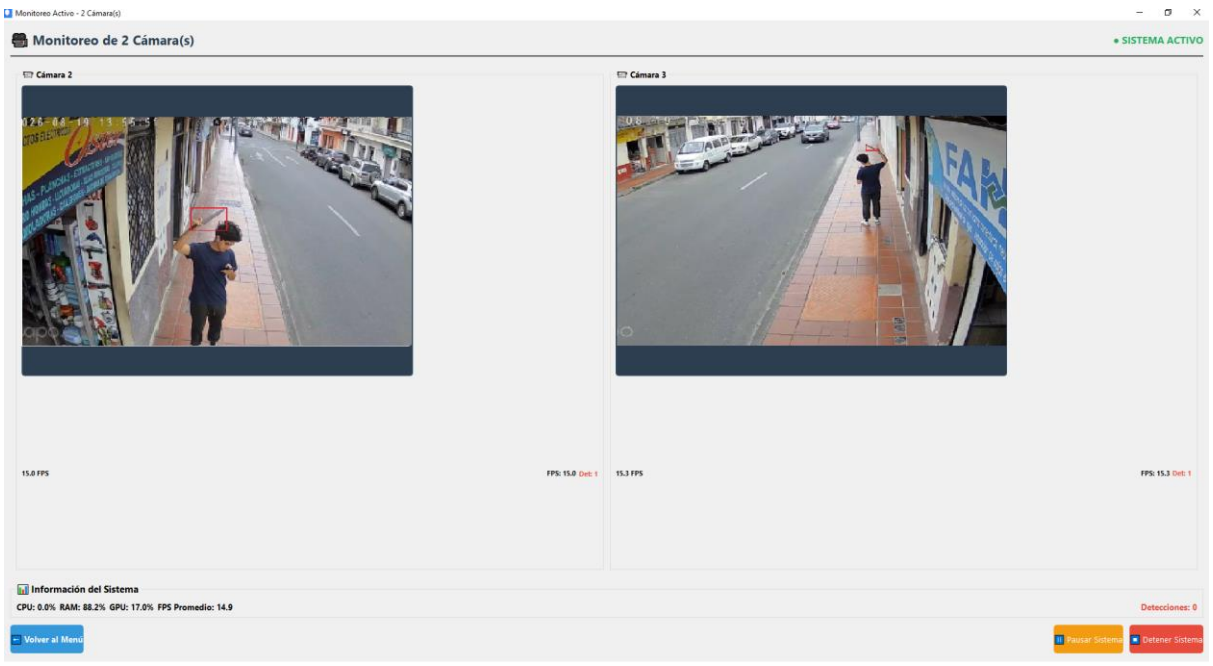
f. Estados y transiciones de controles

Los botones y campos de configuración cambian dinámicamente su estado según la fase operativa del sistema:

- **Inicial:** Solo el botón "Iniciar Monitoreo" está habilitado si hay datos válidos.
- **Monitoreando:** Aparece el botón "Detener Monitoreo", los campos de entrada se desactivan.
- **Detenido:** Se habilitan los botones "Reiniciar" y "Regresar".
- **Error:** Se muestran indicadores visuales (mensajes, colores) notificando fallos operativos.

Figura 29:

Interfaz gráfica del sistema de detección



Nota. Elaboración propia.

g. Monitoreo de recursos del sistema

h. Arquitectura del sistema de monitoreo

El monitoreo del rendimiento se integra visualmente en la ventana de detección. Este panel permite verificar en tiempo real el consumo de recursos, lo cual es crucial para garantizar la estabilidad del sistema en operaciones continuas.

Componentes de monitoreo implementados

- **CPU:** Porcentaje de carga procesador, medido con psutil.
- **RAM:** Memoria utilizada y total disponible, también mediante psutil.
- **GPU:** Utilización de la tarjeta gráfica, reportada por GPUtil.

i. Implementación técnica

La obtención de métricas se realiza con las siguientes bibliotecas:

- psutil para CPU y RAM.

- GPUtil para uso de GPU (maneja errores si el sistema no cuenta con GPU).

Las métricas se actualizan cada segundo, lo que permite un equilibrio entre precisión y bajo impacto en el rendimiento.

j. Visualización adaptativa y niveles

La interfaz visualiza el estado de los recursos mediante tarjetas ubicadas en la parte superior derecha de la ventana. Cada métrica tiene colores adaptativos:

- Verde: Uso bajo (< 50%)
- Naranja: Uso moderado (50%-80%)
- Rojo: Uso alto (> 80%)

k. Posicionamiento y diseño adaptativo

El panel de métricas está anclado al borde derecho de la interfaz, reposicionándose dinámicamente según el tamaño de la ventana para mantener una visualización accesible en todos los dispositivos.

l. Beneficios funcionales

- **Prevención de errores:** Identificación de cuellos de botella en tiempo real.
- **Ajustes dinámicos:** Posibilidad de intervenir con acciones correctivas ante sobrecarga.
- **Soporte para análisis posterior:** Las métricas pueden ser registradas en archivos de log para auditorías y análisis de rendimiento futuro.

Este subsistema de monitoreo convierte la interfaz en una herramienta robusta no solo para detección, sino también para análisis operativo, optimización y mantenimiento proactivo del entorno.

3.4. Desarrollo del backend y API REST (Django)

3.4.1. Almacenamiento de alertas e imágenes

a. Modelo de datos para alertas

El sistema de detección de armas implementa su lógica de almacenamiento utilizando Django como framework principal del backend. La estructura de datos está definida en el archivo models.py dentro de la aplicación alertuploadREST, donde el modelo central UploadAlert representa cada evento registrado por el sistema.

Este modelo establece una relación directa con los usuarios autenticados mediante tokens, y permite asociar cada alerta con su respectivo receptor, ubicación y evidencia visual.

b. Estructura del modelo UploadAlert

Tabla 21:

Campos del modelo UploadAlert

Campo	Tipo de Dato	Descripción	Función en el sistema
image	ImageField	Imagen capturada en el momento de la detección	Almacena evidencia visual con nombre aleatorio para mayor seguridad
userID	ForeignKey(Token)	Identificador del token del usuario	Relaciona la alerta con el usuario autenticado
alertReceiver	CharField(200)	Correo electrónico del receptor	Define el destino de la notificación
location	CharField(200)	Lugar donde ocurrió la detección	Permite ubicar geográficamente el evento
dateCreated	DateTimeField	Fecha y hora de creación del registro	Generado automáticamente como timestamp de la alerta

Nota. Elaboración propia.

c. Gestión de archivos y almacenamiento en la nube

El campo image del modelo utiliza un sistema personalizado de nombres aleatorios basado en UUID4, implementado mediante la función scrambleUploadedFilename(). Esta función garantiza que cada imagen se almacene con un identificador único, evitando conflictos de nombres y reduciendo el riesgo de acceso no autorizado (véase el apartado 9.5 del Manual del Programador, Anexo F).

Adicionalmente, se mantiene la extensión original del archivo (.jpg, .png, etc.) para asegurar la compatibilidad con sistemas de visualización y descarga.

d. Almacenamiento en la nube con AWS S3

Las imágenes se almacenan de forma segura en la nube utilizando Amazon S3, mediante el backend PublicMediaStorage() integrado con django-storages. Esta arquitectura permite:

- **Escalabilidad automática:** Soporte para miles de imágenes sin afectación al rendimiento
- **Alta disponibilidad y redundancia:** SLA de durabilidad del 99,999999999 % (11 nueves)
- **Acceso público controlado:** Las URLs generadas están protegidas y expiran automáticamente

e. Configuración de base de datos

El sistema emplea PostgreSQL como motor de base de datos relacional, configurado para entornos de producción sobre la plataforma Render. Las configuraciones específicas incluyen:

- **Conexiones seguras (SSL requerido):** Garantiza la confidencialidad y autenticidad de las transacciones

- **Parámetros de keepalive:** Previenen la desconexión automática de sesiones largas
- **Timeouts optimizados:** Respuesta oportuna ante posibles caídas de conexión (máximo 5 segundos)
- **Escalabilidad concurrente:** Capacidad para manejar múltiples solicitudes simultáneas sin pérdida de rendimiento

Gracias a esta arquitectura de almacenamiento robusta y segura, el sistema garantiza la trazabilidad completa de cada alerta registrada, asegurando la integridad de los datos y la disponibilidad de la evidencia visual para su posterior análisis y visualización desde el dashboard web.

3.4.2. Gestión de usuarios y autenticación

El sistema de gestión de usuarios y autenticación fue desarrollado utilizando Django y Django REST Framework, implementando un esquema seguro basado en tokens de acceso para proteger la comunicación cliente-servidor y restringir el acceso a recursos sensibles del sistema.

a. Sistema de Autenticación por Tokens

Cada usuario recibe su token en el momento del registro, sin intervención manual. Con ese token, el cliente PyQt5 firma cada petición HTTP y las credenciales viajan una sola vez —en el login—, no en cada transacción. La generación corre sobre señales de Django: la señal post_save del modelo de usuario dispara la creación del token apenas se guarda el registro, como muestra la Figura 30.

Figura *30*:

Generación automática de tokens con post_save

```
@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def create_auth_token(sender, instance=None, created=False, **kwargs):
    if created:
        Token.objects.create(user=instance)
```

Nota. Elaboración propia.

El token se obtiene desde un único endpoint:

b. Endpoints de Autenticación

El endpoint principal para la obtención del token es:

- **URL:** /api/get_auth_token/
- Método: POST
- **Parámetros:** username, password
- **Respuesta:** Objeto JSON con el token de autenticación.

La configuración de la URL se incluye en el archivo urls.py de la aplicación REST, como se detalla en la Figura 31:

Figura 31:

Configuración del endpoint de autenticación

```
# Configuración en urls.py
url(r'^get_auth_token/$', rest_framework_views.obtain_auth_token, name='get_auth_token'),
```

Nota. Elaboración propia.

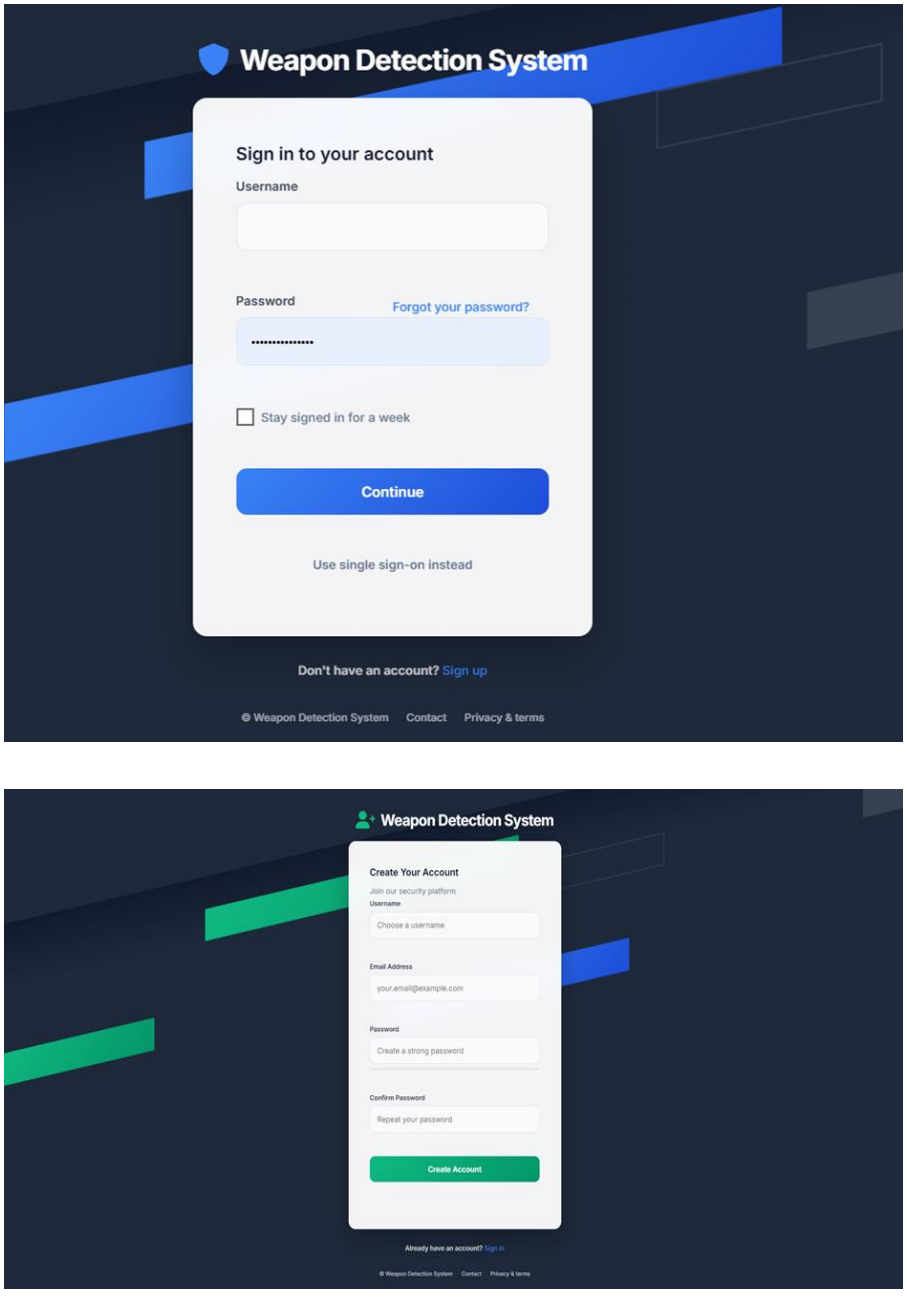
c. Registro de Usuarios

El registro extiende el UserCreationForm de Django con una validación extra: el correo electrónico debe ser único. Sin ese control, dos cuentas podían compartir email y las alertas terminaban en un receptor ambiguo; el formulario extendido rechaza el

duplicado antes de tocar la base de datos (véase el apartado 9.7 del Manual del Programador, Anexo F). La Figura 32 muestra las pantallas de login y registro.

Figura 32:.

Formularios gráficos de inicio de sesión y registro.



Nota. Elaboración propia.

d. Vistas de Autenticación y Seguridad

- **Login:** Verifica las credenciales contra la base de datos. Si coinciden, redirige al dashboard; si no, devuelve el formulario con el mensaje de error correspondiente.
- **Registro:** Crea la cuenta tras validar que el email no exista y deja al usuario con sesión iniciada, sin pedirle un segundo login.
- **Logout:** Invalida la sesión activa y devuelve al formulario de acceso con un mensaje de confirmación.

Ninguna ruta del dashboard queda expuesta: el middleware de autenticación intercepta toda petición sin sesión válida y la redirige al login.

e. Envío de Notificaciones por Correo Electrónico

Una vez autenticado el usuario y generado el token, el sistema emite alertas que se envían de forma automática por correo electrónico al receptor configurado, como se ejemplifica en el Anexo A.

3.4.3. Envío de notificaciones por correo electrónico

a. Flujo de Notificaciones Automáticas

El sistema de detección de armas incluye una funcionalidad crítica de notificación que permite alertar de forma inmediata al contacto definido por el usuario. Este proceso se realiza de manera asíncrona y segura, garantizando el envío de un mensaje personalizado por correo electrónico al receptor configurado.

b. Proceso de Identificación y Envío

El proceso comienza al completarse la validación del modelo y generarse una alerta. A partir de allí, una función verifica con una expresión regular que el receptor sea una dirección de correo electrónico válida; si lo es, se dispara en un hilo independiente el envío del correo (Anexo B).

c. Sistema de Notificaciones por Email.

Para el envío de correos electrónicos, el sistema utiliza la API HTTP de Brevo (Sendinblue). Render bloquea el SMTP saliente por el puerto 587, de modo que el envío se resuelve como una petición HTTPS al endpoint https://api.brevo.com/v3/smtp/email. La configuración se realiza mediante variables de entorno para proteger las credenciales (Véase Anexo C). El cuerpo del mensaje se genera dinámicamente e incluye un enlace directo al detalle de la alerta.

- **Proveedor:** Brevo (Sendinblue), API HTTP transaccional
- **Puerto:** 443 (HTTPS)
- **Autenticación:** clave BREVO_API_KEY almacenada en variable de entorno
- **Remitente:** dirección DEFAULT_FROM_EMAIL verificada como sender en Brevo

d. Armado del mensaje y respaldo ante fallos.

Antes del envío, el servidor arma los datos de la alerta: identificador, fecha y hora, ubicación y enlace al detalle. Con ellos genera dos versiones del mensaje, una en HTML y otra en texto plano de respaldo. Si el envío del correo enriquecido falla por cualquier motivo, un segundo hilo reintenta con la versión simple, de modo que un error de plantilla nunca deja al operador sin aviso (Véase Anexo D).

e. Generación de Mensajes de Alerta.

El contenido del mensaje es generado automáticamente con un enlace personalizado que dirige al usuario directamente a la página de detalle de la alerta registrada en el sistema (véase el apartado 9.8 del Manual del Programador, Anexo F).

f. Características del Sistema de Notificación.

- **Procesamiento asíncrono:** El uso de hilos permite no bloquear la ejecución principal ni afectar la experiencia del usuario.

- **Validación robusta:** Se valida el formato del correo mediante expresiones regulares para asegurar entregabilidad.
- **Tolerancia a fallos:** Los envíos están configurados para manejar errores sin interrumpir el resto del sistema.
- **Personalización automática:** Los mensajes generados incluyen el enlace único hacia la evidencia de la alerta.

3.5. Desarrollo del panel web de visualización

3.5.1. Dashboard de alertas en tiempo real

El sistema cuenta con un dashboard web diseñado para la visualización en tiempo real de las alertas generadas por el modelo de detección de armas. Esta interfaz se implementó utilizando Django en el backend y tecnologías web modernas en el frontend, priorizando la usabilidad, el acceso rápido a los datos y la presentación visual clara de las detecciones.

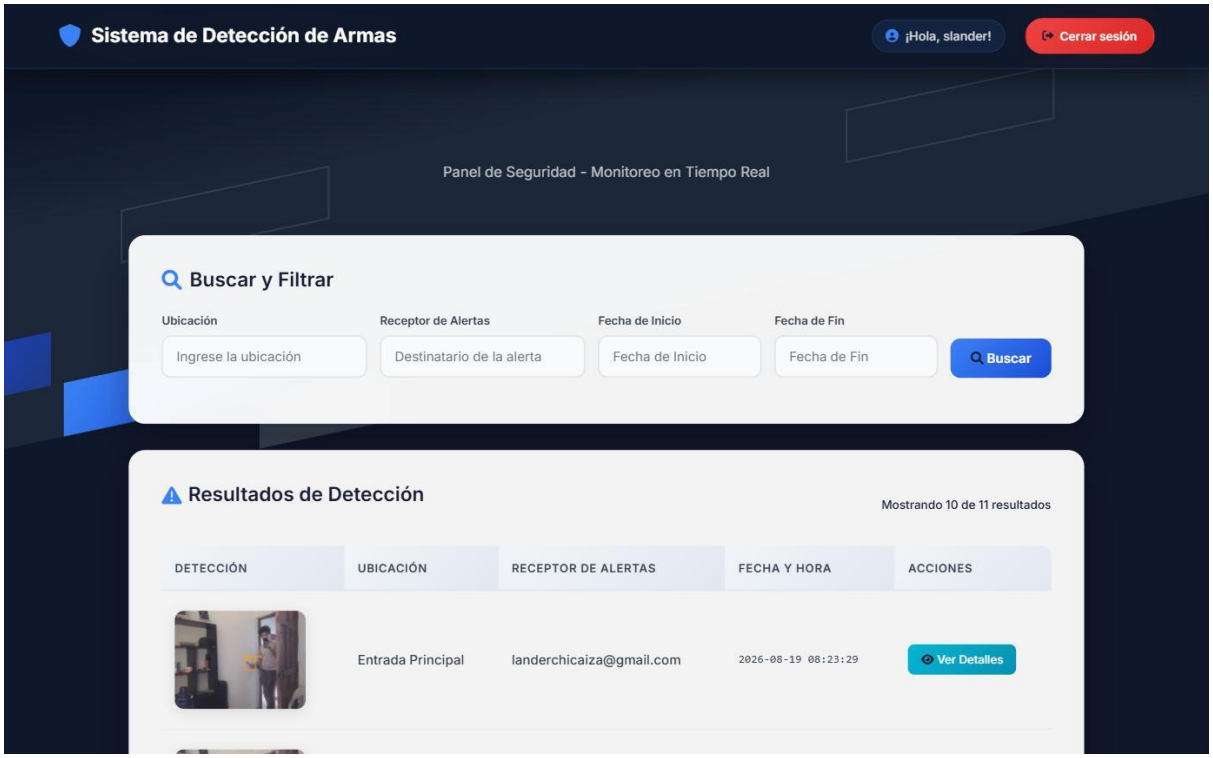
a. Elementos principales del dashboard:

El panel reparte la pantalla en cuatro zonas, cada una con una función distinta:

- **Barra de navegación superior: muestra** el nombre del sistema ("Weapon Detection System"), los botones de login y logout y el nombre del usuario autenticado, de modo que la sesión se abre y se cierra desde cualquier pantalla.
- **Sección de búsqueda y filtros: acota** los resultados por ubicación y por receptor de la alerta mediante campos de texto, y por rango de fechas mediante selectores con calendario. El botón Search aplica el filtro de inmediato (Figura 33).

Figura 33:

Interfaz Aplicación de fechas y filtros

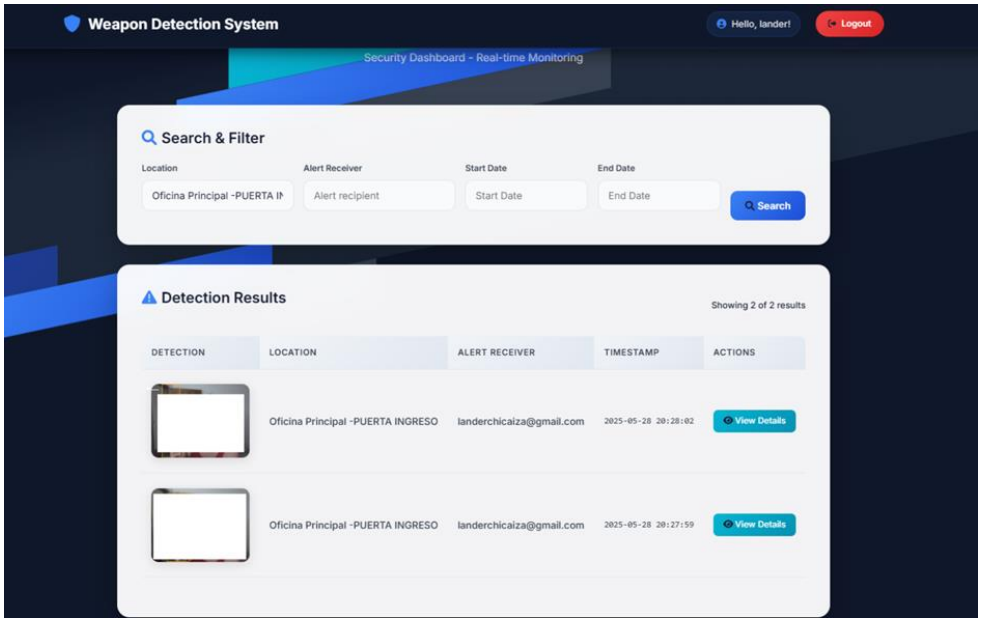


Nota. Elaboración propia.

- **Panel de resultados:** una tabla lista las alertas detectadas. Cada fila trae:
  - Miniatura de la imagen capturada
  - Ubicación geográfica de la detección
  - Receptor del mensaje de alerta
  - Timestamp de creación de la alerta
  - Botón View Details para acceder a información completa.

Figura 34:

Aplicación de filtro activado

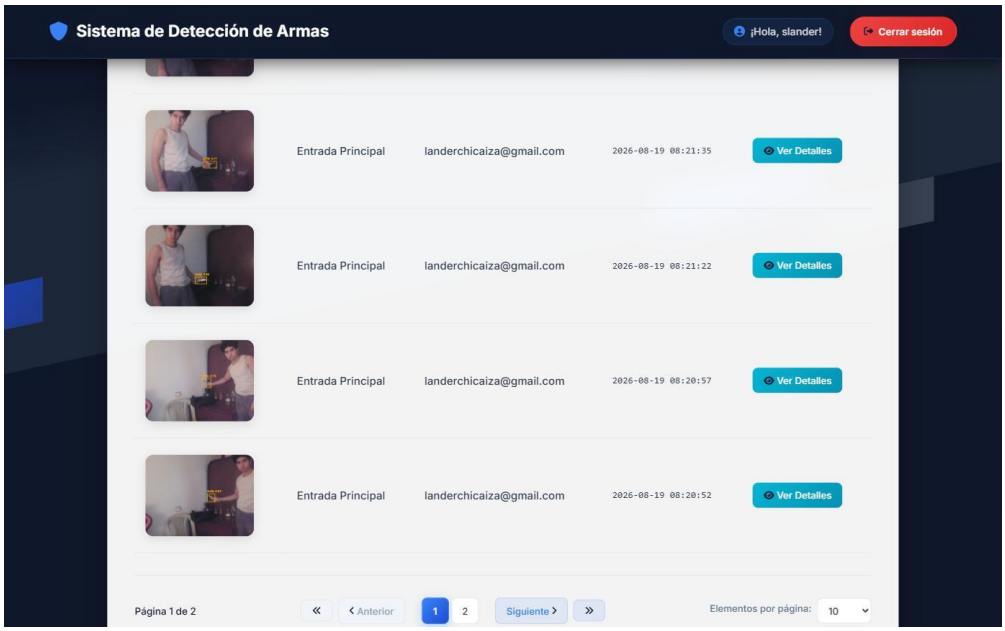


Nota. Elaboración propia.

- **Sistema de paginación: recorre las** páginas de resultados con los botones First, Previous, Next y Last, informa cuántos registros hay en total y deja elegir cuántos se muestran por página.

Figura 35:

Interfaz paginación en el panel de alertas



Nota. Elaboración propia.

b. Elementos visuales destacados:

- Los íconos de Font Awesome refuerzan el significado de cada botón.
- El diseño utiliza azules (#3b82f6) y grises (#374151).
- Badges como “ACTIVE THREAT DETECTED” refuerzan el estado de cada alerta.

c. Características de usabilidad integradas:

- **Diseño responsivo:** Optimizado para pantallas de escritorio y móviles.
- **Filtros dinámicos:** Aplicación inmediata sin necesidad de recargar la página.
- **Retroalimentación visual:** Indicadores de carga y mensajes de estado del sistema.
- **Navegación amigable: jerarquía visible,** breadcrumbs en cada vista y acciones con etiqueta propia.

Con esos elementos, el administrador supervisa desde una sola pantalla todos los eventos registrados y llega a cualquiera de ellos en pocos clics.

3.5.2. Filtros de búsqueda y detalles individuales

a. Sistema de Filtros de Búsqueda

El filtrado del panel web se apoya en la clase DetectionFilter, heredada de django_filters.FilterSet. El usuario combina varios criterios en una misma consulta y el filtro los aplica sobre el queryset de alertas, sin exponer los campos que no corresponden (Tabla 22).

Tabla 22

Campos de búsqueda disponibles en el sistema.

Campo	Tipo de Filtro	Descripción
location	Texto (icontains)	Permite localizar alertas por nombre parcial de la ubicación (ej. "Entrada", "Oficina").
alertReceiver	Texto (icontains)	Filtra alertas por el correo electrónico del receptor.
startDate	Fecha (mayor o igual que)	Define el inicio del rango temporal de búsqueda (con interfaz de calendario).
endDate	Fecha (menor o igual que)	Define el límite superior del rango de fechas para detecciones.

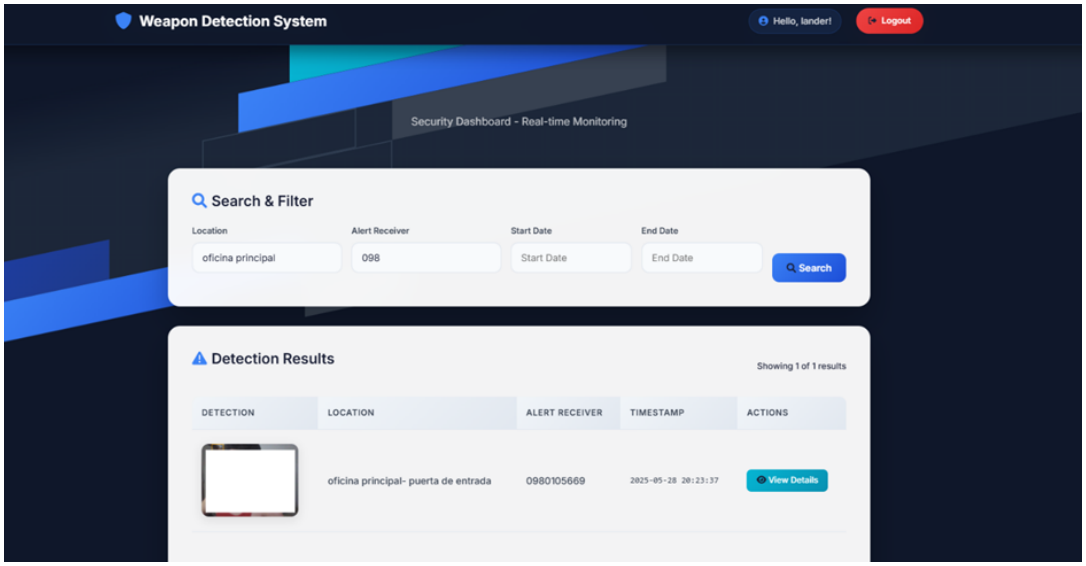
Nota. Elaboración propia.

b. Características técnicas de los filtros:

- **Exclusiones configuradas: cuatro campos quedan fuera** del filtro —customer, userID, image y uuid— para que la consulta no exponga datos que el usuario no necesita.
- **Persistencia de estado: los** parámetros viajan en la URL, así que una búsqueda se comparte o se retoma con solo copiar la dirección.
- **Combinación lógica:** Los filtros se aplican utilizando el operador lógico AND, lo cual permite búsquedas refinadas.
- **Validación de entrada:** Todos los datos de entrada son automáticamente sanitizados para evitar errores o vulnerabilidades.

Figura 36:

Aplicación de filtros por diferentes tipos de búsqueda: ubicación, receptor y fechas



Nota. Elaboración propia.

c. Página de Detalles Individuales

La plantilla alert.html abre una alerta concreta para su revisión posterior: es la vista a la que acude el operador cuando necesita reconstruir un evento ya notificado. Reúne en una sola pantalla la evidencia visual y los metadatos completos del incidente. Sus componentes:

d. Componentes de la vista detallada:

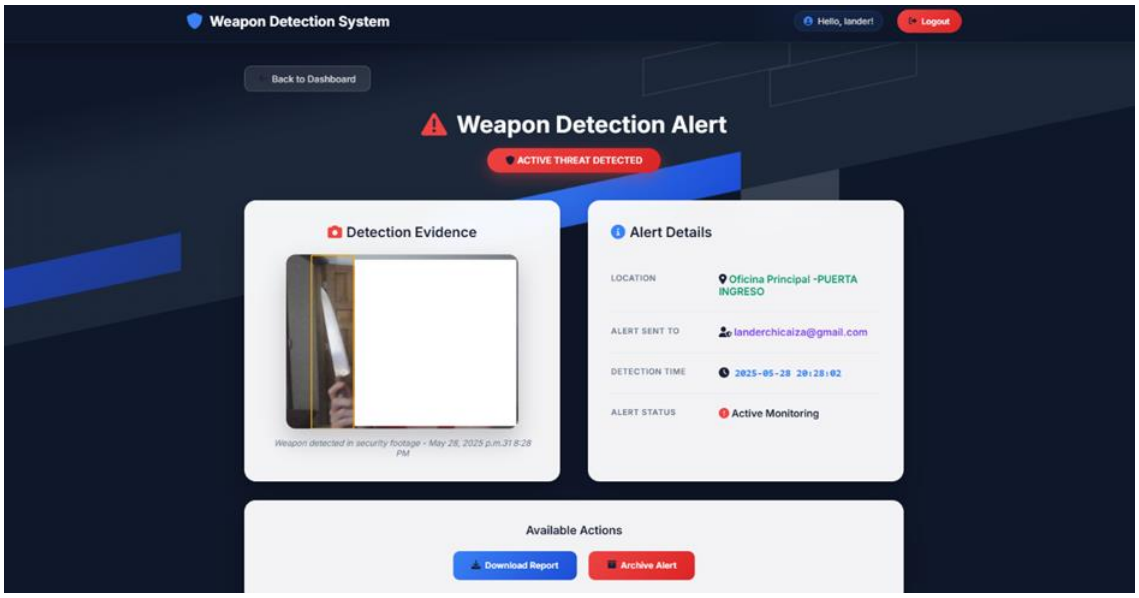
1. Panel de Evidencia Fotográfica:
    - Muestra la imagen capturada durante la detección.
    - Permite visualización ampliada y metadatos básicos como resolución y timestamp.
2. Información del Incidente:
    - **Ubicación:** Descripción completa del lugar donde ocurrió la detección.
    - **Receptor de alerta:** Correo electrónico del destinatario.
    - **Fecha y hora:** Timestamp exacto del evento con zona horaria incluida.

3. Panel de Acciones Disponibles:

    - **Descargar Reporte:** Exporta un documento en formato de documento portátil (PDF) con la imagen y los metadatos.
    - **Archivar Alerta:** el botón muestra una confirmación en pantalla, pero en esta versión el cambio no se guarda en el servidor y la alerta permanece en el listado (véase el apartado 9.12 del Manual del Programador, Anexo F).

Figura 37:

Visualización de detalles individuales de una alerta



Nota. Elaboración propia.

e. Funcionalidades avanzadas del sistema:

- Generación de reportes PDF automáticos, que incluyen:
    - Imagen de evidencia en alta calidad.
    - Metadatos completos de la alerta.
    - Información detallada del incidente.
- **Botones de acción interactiva**: descargar el reporte en PDF y archivar la alerta, con la limitación indicada arriba.

Con esta vista, el sistema deja de ser solo un detector. Cada alerta queda documentada con su imagen y sus metadatos — el respaldo que un local necesitaría ante una denuncia o un proceso legal.

3.6. Validación del sistema

3.6.1. Escenarios de prueba y resultados funcionales

Metodología de Validación

La validación funcional se ejecutó como una sesión formal de aseguramiento de calidad (Quality Assurance, QA), estructurada según el estándar IEEE 829-2008 y registrada con el identificador QA-WDS-2026-08-S01. El autor la corrió el 19 de agosto de 2026 sobre el commit 410d1ea de la rama principal, con el sistema ya desplegado en su configuración definitiva y apuntando al servidor en producción. Ningún módulo se dio por bueno sin ejercitarlo: cada caso corrió con las cámaras activas, el usuario interactuando con la interfaz y las alertas disparándose de verdad. El Anexo G recoge el informe completo con su evidencia fotográfica.

El entorno fue el equipo cliente del protocolo de instrumentación: AMD Ryzen 5 4600H, 16 GB de memoria y GPU NVIDIA GTX 1650 sobre Windows 10. El sistema arrancó desde el repositorio con python main.py, con el umbral de confianza declarado en settings.ini. Las cámaras fueron dos TP-Link Tapo C310, una por local. El servidor corrió sobre Render con Django 4.2.16 y PostgreSQL, y el correo salió por la API HTTP de Brevo.

El plan de pruebas QA define 21 casos, todos ejecutados en esta sesión. Cubren las rutas que, de fallar, dejarían inoperante la cadena de notificación: autenticación y gestión de sesión, configuración, gestión de cámaras, motor de inferencia, detección y evidencia, API y persistencia, panel de control web y canal de notificación por correo. Todos los casos ejecutados se superaron y ninguno reveló un fallo que impidiera la operación. La Tabla 23 recoge la matriz completa.

Tabla 23*:*

Matriz de las pruebas funcionales ejecutadas.

Caso	Conjunto	Descripción	Resultado	Evidencia
CP-AUT-001	TDS-01	Registro de usuario y generación automática de token	PASA	Token asociado en el panel de administración
CP-AUT-003	TDS-01	Registro con datos duplicados y validación de formulario	PASA	Dos cuentas creadas sin colisión
CP-AUT-006	TDS-01	Login con credenciales correctas	PASA	Saludo «¡Hola, qa_operador_a1!»
CP-AUT-007	TDS-01	Login con credenciales incorrectas	PASA	«El nombre de usuario o la contraseña es incorrecta»
CP-AUT-011	TDS-01	Cierre de sesión y redirección	PASA	Log: POST /logout/ → 302 → GET /login/
CP-AUT-012	TDS-01	Acceso sin sesión activa	PASA	Redirección automática a /login/
CP-CFG-006	TDS-02	Umbral de confianza aplicado desde settings.ini	PASA	Valor confirmado en detecciones reales
CP-CFG-009	TDS-02	Lectura de configuración en el arranque en frío	PASA	settings.ini leído al iniciar el cliente
CP-CAM-011	TDS-04	Gestor de cámaras: alta, edición, importación y exportación	PASA	Gestor con cinco cámaras registradas
CP-CAM-012	TDS-04	Monitoreo simultáneo de dos cámaras (Local A y Local B)	PASA	Captura de monitoreo multicámara activo

CP-ENG-001	TDS-06	Carga única del modelo al arranque	PASA	Log de consola: una sola carga por sesión
CP-DET-006	TDS-08	Detección guardada localmente con etiqueta y confianza	PASA	Archivo de evidencia del 19/08/2026, confianza 0.81
CP-API-001	TDS-09	Persistencia de alertas en el backend mediante token	PASA	100 registros verificados en Upload alerts mediante autenticación por token; identificador de usuario de qa_operador_a1 asociado a cada registro
CP-API-014	TDS-09	Rechazo de método HTTP no permitido en /api/images/	PASA	GET → 405, Allow: POST, OPTIONS
CP-DASH-001	TDS-11	Redirección a login sin sesión activa	PASA	Ventana privada redirigida a /login/
CP-DASH-002	TDS-11	Listado de alertas con paginación	PASA	«Mostrando 10 de 136 resultados»
CP-DASH-003	TDS-11	Filtros por ubicación, receptor y rango de fechas	PASA	Panel «Buscar y Filtrar» operativo
CP-DASH-004	TDS-11	Vista de detalle de alerta individual	PASA	Evidencia, ubicación, receptor, hora y acciones
CP-COR-001	TDS-13	Envío del correo de alerta por la API HTTP de Brevo	PASA	Correo recibido en la bandeja del receptor
CP-COR-002	TDS-13	Consistencia entre alerta, backend y correo	PASA	Identificador 151 coincidente en las tres fuentes
CP-COR-003	TDS-13	Formato y contenido del correo	PASA	«Azuay entre Juan José Peña y Olmedo», 19/08/2026 16:37:47

Nota. Elaboración propia.

3.6.2. Resultados consolidados por conjunto de pruebas.

Autenticación y gestión de sesión (TDS-01): seis de seis. El token de Django REST Framework se genera y queda asociado al usuario que lo solicita, y cualquier

intento de acceso sin sesión activa termina en el formulario de login. El manejo de credenciales inválidas devuelve el mensaje esperado, en español.

Configuración (TDS-02): dos de dos. El cliente lee settings.ini en el arranque en frío y aplica el umbral de confianza, valor que se confirmó después en las detecciones reales de la sesión.

Gestión de cámaras (TDS-04): dos de dos. El gestor admite alta, edición, exportación e importación de configuraciones, y el monitoreo simultáneo de las dos cámaras se sostuvo sin degradación visible de la interfaz.

Motor de inferencia (TDS-06) y detección y evidencia (TDS-08): uno de uno en cada conjunto. El modelo se carga una sola vez por sesión de worker y la detección se guarda en disco con su etiqueta y su nivel de confianza.

API y persistencia (TDS-09): dos de dos. Cien alertas quedaron registradas en el backend mediante autenticación por token, y el endpoint rechaza con 405 los métodos que no admite.

Panel de control web (TDS-11): cuatro de cuatro. Paginación, filtros por ubicación, receptor y fechas, y vista de detalle con evidencia y acciones asociadas.

Canal de notificación por correo (TDS-13): tres de tres. El correo llegó a la bandeja del receptor con la ubicación y la marca temporal correctas.

El valor de la sesión no está solo en la tasa de éxito. Cada alerta se contrastó contra tres fuentes independientes: el log de consola del servidor, el panel de administración de Django y la bandeja de correo del destinatario. Que las tres coincidan en identificador, ubicación y marca temporal es lo que sostiene la validez de los tiempos que reporta el apartado siguiente.

3.6.3. Pruebas de rendimiento: tiempo de detección, envío y consulta

a. Protocolo de validación e instrumentación de tiempos (T1–T4)

Antes de reportar resultados conviene dejar por escrito bajo qué condiciones se midió y cómo se capturó cada tiempo, de modo que otro establecimiento pueda repetir el ejercicio sin depender del autor. El protocolo tiene dos partes. La primera fija las condiciones mínimas de operación; la segunda, el procedimiento de instrumentación de marcas de tiempo.

Condiciones mínimas de operación: equipo cliente de gama media —en estas pruebas, AMD Ryzen 5 4600H, 16 GB de RAM y GPU NVIDIA GTX 1650—, conexión a internet estable y umbral de confianza fijado en 0,75. Hace falta además al menos una cámara declarada en cameras.json, sea IP sobre RTSP o webcam USB. El servidor opera alojado en Render con base de datos PostgreSQL; el canal de notificación es el correo electrónico por la API HTTP de Brevo. Ninguna de estas condiciones se supone: todas se verifican antes de iniciar la serie de mediciones.

Procedimiento de instrumentación: el cliente registra en el archivo inference_metrics.csv una marca de tiempo por evento (timestamp, camera_id, tiempo de inferencia en milisegundos y tiempo en cola), y el backend añade las suyas al recibir la alerta y al despachar el correo. Con esas marcas se reconstruyen los cuatro tramos de la cadena: T1, de la captura del cuadro a la detección positiva; T2, de la detección a la recepción en el servidor; T3, de la recepción al despacho de la notificación; y T4, del despacho a la llegada de la alerta al dispositivo del receptor.

La medición corrió primero en un entorno controlado y se completó con la operación real en dos establecimientos comerciales del centro de Loja, del 20 al 31 de julio de 2026. Cada local usó sus propias cámaras y su propia red durante todo el período, sin que el autor estuviera presente.

b. Metodología de Evaluación

El rendimiento se midió sobre 100 ciclos completos. Cada ciclo recorre la cadena entera: detección local en el cliente, registro en el backend y entrega del aviso por correo al receptor configurado. Nadie midió etapas sueltas, porque lo que importa es el tiempo que ve el usuario final: del arma frente a la cámara al mensaje en la bandeja. El entorno de pruebas tuvo estas especificaciones:

- **Hardware:** AMD Ryzen 5 4600H, 16 GB de RAM, NVIDIA GTX 1650.
- **Red:** Fibra óptica simétrica de 1 Gbps.
- **Escenarios:** 25 detecciones de pistola, 25 de cuchillo y 50 frames sin armas.

Tabla 24:

Resultados de rendimiento del sistema.

Métrica	Tiempo Promedio	Mínimo	Máximo	Desviación Estándar
Inferencia y guardado de evidencia (T1)	69,8 ms	63,1 ms	78,7 ms	±3,3 ms
Envío al servidor (T2)	1.859,1 ms	1.050 ms	3.419 ms	±615,6 ms
Procesamiento en el backend y despacho del correo (T3)	447,3 ms	302 ms	579 ms	±85,1 ms
Entrega al destinatario (T4)	3.183,7 ms	2.252 ms	4.089 ms	±575,2 ms
Proceso completo (T1 a T4)	5.559,8 ms	4.008,5 ms	7.697,5 ms	±854,2 ms
Consulta en dashboard (10 alertas)	312 ms	234 ms	467 ms	±42 ms
Carga de vista detalle	623 ms	498 ms	834 ms	±89 ms

Nota. Elaboración propia.

La suma de los cuatro tramos sitúa el tiempo total promedio, desde la detección del arma hasta la entrega del correo, en 5.559,8 ms (5,6 s).

La sesión de aseguramiento de calidad aportó una medición independiente del mismo trayecto. El autor extrajo del log de consola del servidor los tiempos de las peticiones POST al endpoint /api/images/ de cuarenta alertas reales procesadas el 19 de agosto de 2026: mínimo de 1,03 s, máximo de 3,75 s y valor típico entre 1,5 y 2,1 s (Anexo G, apartado 4).

Ese rango no contradice la Tabla 24, porque mide un tramo más corto. La petición POST responde en cuanto el backend guarda la alerta y lanza el envío del correo en un hilo aparte. El tiempo del log abarca entonces T2 y el procesamiento del servidor, pero no el despacho hacia Brevo, que ocurre después de devolver la respuesta al cliente.

Tabla 25:

Distribución percentil de tiempos de respuesta

Proceso	**P50 (Mediana)**	**P90**	**P95**	**P99**
Inferencia (T1)	70,1 ms	73,3 ms	75,5 ms	77,3 ms
Envío al servidor (T2)	1.781 ms	2.686 ms	3.094 ms	3.308 ms
Backend y despacho (T3)	444 ms	562 ms	572 ms	579 ms
Entrega al destinatario (T4)	**3.093 ms**	4.010 ms	4.044 ms	4.089 ms
Proceso completo (canal correo)	**5.683 ms**	6.617 ms	6.834 ms	7.595 ms

Nota. Elaboración propia.

c. Métricas del modelo en operación

Los 100 ciclos sirven además para contrastar el modelo contra sus métricas de entrenamiento. De los 100 escenarios, 50 contenían un arma —25 con pistola y 25 con

cuchillo— y 50 no. El autor clasificó cada ciclo como verdadero positivo, falso positivo, falso negativo o verdadero negativo, según lo que el sistema alertó frente a lo que había en escena.

Resultados de la clasificación: 42 verdaderos positivos, 9 falsos positivos, 8 falsos negativos y 41 verdaderos negativos. Esos conteos dan una precisión operativa de 82,4 %, un recall de 84,0 % y un F1-score de 83,2 %. La exactitud global, que pondera los aciertos sobre el total de ciclos, llegó a 83,0 %. Las dos clases rindieron igual: 21 de 25 escenarios con pistola y 21 de 25 con cuchillo terminaron en detección confirmada.

En entrenamiento el modelo reportó 90,3 % de precisión, 89,4 % de recall y 91,2 % de mAP@0.5, equivalentes a un F1-score de 89,8 %. La Tabla 26 contrasta ambos entornos. El mAP no se recalcula en operación: exigiría anotar el bounding box cuadro a cuadro, y eso queda fuera del alcance de la prueba de campo. La distancia entre laboratorio y operación quedó en 7,9 puntos porcentuales de precisión, 5,4 de recall y 6,6 de F1-score. La caída es moderada y apunta en una sola dirección. El modelo conserva su capacidad de discriminación fuera del entorno controlado, pero paga el costo de trabajar con ángulos de cámara fijos, iluminación cambiante y objetos cotidianos que nunca estuvieron en el conjunto de entrenamiento.

Tabla 26:

Operación de sistema bajo 100 ciclos.

Métrica	**Entrenamiento**	**Operación real (100 ciclos)**
Precisión	90,3 %	82,4 %
Recall	89,4 %	84,0 %
F1-score	89,8 %	83,2 %
mAP@0.5	91,2 %	No aplica

Nota. Elaboración propia.

Los errores no se repartieron al azar. Los ocho falsos negativos comparten causa: el ángulo de la cámara dejó el arma parcialmente ocluida —cuatro casos con cuchillo y

cuatro con pistola—, casi siempre cuando el sujeto la sostenía por debajo de la línea del mostrador. Los nueve falsos positivos se concentran en cuatro objetos. Un taladro negro de mango curvo disparó cinco alertas; una billetera oscura abierta, dos; un escáner de código de barras y un teléfono sostenido en ángulo, una cada uno. El detector no confunde cualquier objeto: confunde siluetas oscuras con mango. Ese diagnóstico orienta las recomendaciones del Capítulo IV mucho mejor que una tasa global de error.

d. Análisis de Factores que Afectan el Rendimiento

Los principales cuellos de botella detectados corresponden a:

1. Entrega del correo al destinatario (57,3 % del total):
    - Tiempo promedio de 3.183,7 ms, medido sobre las cabeceras del correo recibido
    - Latencia variable dependiente del proveedor y red
2. Envío de la imagen al servidor (33,4 %):
    - Tamaño promedio: 245 KB
    - Compresión JPEG aplicada
    - Altamente dependiente del ancho de banda disponible
3. Procesamiento en el backend y despacho del correo (8,0 %):
    - Validación, almacenamiento en PostgreSQL y subida a S3
4. Inferencia local (1,3 %):
    - YOLOv8 con cache de modelo activo. En la sesión del Anexo G la inferencia corrió sobre CPU por la incidencia INC-15, no sobre GPU.

e. Optimizaciones Aplicadas

Para mantener los tiempos dentro del umbral objetivo (<7 s), se implementaron las siguientes mejoras:

- **Compresión de imágenes:** Reducción del 60% sin pérdida significativa
- **Pool de conexiones HTTP:** Disminución de latencia en transmisión
- **Cache de modelo:** Eliminación de tiempos de carga por frame
- **Procesamiento asincrónico:** Envío de alertas en hilos separados para no bloquear la detección.

Conviene anotar una limitación sobre la trazabilidad de esta medición. La instancia de PostgreSQL utilizada en Render se recicla de forma periódica, por lo que los registros generados durante la operación de julio ya no están disponibles en línea. La evidencia de esa fase la constituyen los archivos de imagen guardados localmente por el cliente, las capturas del panel de administración y los correos recibidos, todos ellos conservados por el autor. Los identificadores de alerta que aparecen en el Anexo G corresponden a una numeración posterior al reciclado de la base y no deben leerse como continuación de los de julio.

3.7. Validación con usuarios: retroalimentación recibida

a. Metodología de Validación

La validación de usabilidad se realizó con los responsables de los dos establecimientos comerciales de Loja donde operó el sistema. No fueron sesiones de laboratorio: durante doce días, del 20 al 31 de julio de 2026, el sistema corrió de forma continua sobre las cámaras de cada local y, dentro de ese período, se ejecutaron cincuenta ciclos controlados por establecimiento. En la mitad de esos ciclos una persona designada portaba el objeto de prueba frente a la cámara; la otra mitad servía como control. Ninguno de los dos participantes presenció un hecho delictivo real, y las entrevistas se redactaron para reflejar esa condición.

Los ensayos corrieron sin aviso previo al participante, para que la reacción medida correspondiera a una jornada de trabajo normal. Al cierre del período el autor aplicó una entrevista estructurada de veintiuna preguntas en cinco bloques: desarrollo de los ensayos, observación del sistema, manejo cotidiano, valoración numérica de seis aspectos en escala de 1 a 10, y proyección de lo observado hacia una situación real. El guion completo consta en el Anexo H.

Tabla 27:

Perfiles de usuarios participantes

Participante	**Perfil**	**Experiencia**	**Contexto de Aplicación**
P1 (Local A)	Propietario del negocio	12 años al frente	Comercio de cajas de medidores de agua y luz. Atención de 08:00 a 18:00. Una cámara exterior en la entrada.
P2 (Local B)	Técnico y encargado de taller, hijo del propietario	20 años de operación del negocio familiar	Servicio técnico y reparación de electrodomésticos. Atención de 08:30 a 18:30. Una cámara interior sobre el mostrador y el acceso.

Nota. Elaboración propia.

b. Retroalimentación Cualitativa

b.1. Participante P1 — propietario del Local A

P1 estuvo presente solo en una parte de los ciclos y describió el sistema como un vigilante que trabaja sin supervisión. Destacó que el aviso llegaba al teléfono sin que él tuviera que mirar nada, y que la fotografía adjunta bastaba para identificar a la persona sin abrir la cámara. Su reacción típica al recibir la alerta se situó entre cuatro y diez segundos, el tiempo de sacar el teléfono del bolsillo. Sobre el comportamiento del

detector, observó menor precisión en horarios nocturnos y ninguna afectación por la luz solar directa, que la inclinación de la cámara hacia el suelo neutraliza.

«El sistema me quita de encima la labor de estar pendiente todo el tiempo de mis cámaras; solo tengo que estar pendiente de cuando llega una alerta, y cuando llega, sé que va a estar la fotografía de la persona que ingresó.»

b.2. Participante P2 — encargado del taller del Local B

P2 presenció la mayoría de los ciclos, siempre mientras trabajaba en reparaciones y de espaldas al acceso. Ese detalle condiciona toda su valoración: en un taller nadie puede mirar un monitor, de modo que el canal de aviso importa más que la interfaz. Tardaba entre cinco y diez segundos en limpiarse las manos y revisar el teléfono. A diferencia de P1, reconoció dificultades iniciales con el panel web por su escaso manejo de plataformas informáticas, y las resolvió una vez que le indicaron dónde consultar las fotografías. También reportó variaciones de imagen entre la mañana, con sol directo por la puerta, y la tarde, bajo iluminación fluorescente.

«En un taller uno pasa concentrado en las reparaciones y de espaldas a la entrada; el sistema nos da tranquilidad porque nos avisa directo al teléfono con la foto sin necesidad de estar pegados a una pantalla.»

Ambos coincidieron en un punto que atañe al diseño del sistema: los dos observaron alertas disparadas por objetos que no eran armas. P1 las atribuyó a objetos presentes en la escena durante los ciclos de control; P2, a herramientas metálicas de gran tamaño y a movimientos rápidos cerca del mostrador. También coincidieron en que hubo ciclos con objeto de prueba en los que la alerta no llegó, y P2 aportó la explicación: la persona entraba de espaldas o el objeto quedaba cubierto por una caja de repuestos. Las cifras exactas de ambos fenómenos proceden del registro del sistema y se reportan en la sección 3.6.2; lo que aportan las entrevistas es la caracterización de las situaciones que los provocaron.

Tabla 28:

Evaluación Consolidada

Aspecto evaluado	P1 (Local A)	P2 (Local B)	Promedio
Facilidad de uso	9/10	7/10	8,0/10
Precisión de detección	8/10	8/10	8,0/10
Velocidad de respuesta	10/10	9/10	9,5/10
Utilidad del panel web	8/10	8/10	8,0/10
Adaptabilidad al local	10/10	9/10	9,5/10
Relación costo-beneficio	8/10	9/10	8,5/10
Puntuación general	9/10	8/10	8,5/10

Nota. La puntuación general corresponde al juicio global que cada participante asignó al sistema, no al promedio aritmético de los seis aspectos, que sería 8,8 para P1 y 8,3 para P2. Elaboración propia.

c. Percepción General del Tiempo de Respuesta

Los dos participantes calificaron de rápido el tiempo de respuesta, que promedió 5,6 segundos desde la detección hasta la llegada del correo. El argumento que dieron fue el mismo, y no se refiere al reloj: ninguno de los dos puede vigilar una pantalla mientras trabaja. P1 atiende a sus clientes; P2 repara electrodomésticos de espaldas a la puerta. Frente a esa condición, 5,6 segundos de espera pesan poco al lado del costo de mantener a alguien mirando cámaras.

La proyección hacia un hecho real matizó esa valoración. Ambos anticiparon un cambio de conducta. P1 llamaría de inmediato al ECU 911 y subrayó que podría hacerlo sin estar en el local, lo que reduce el riesgo para quien esté dentro; P2 buscaría resguardo en la parte trasera del taller antes de avisar a las autoridades. Los dos señalaron, sin que se les preguntara, otro valor del sistema: la fotografía inmediata sirve para identificar

después al autor y para sostener la denuncia. Ninguno afirmó que el aviso permitiría impedir el asalto en curso.

d. Aspectos Comunes Destacados por los Usuarios

La evidencia visual fue lo mejor valorado por ambos. Consultados por un aviso instantáneo sin fotografía, con los detalles después, los dos lo rechazaron por la misma razón: la imagen les deja descartar una falsa alarma antes de moverse. P1 lo llamó «un segundo filtro» sobre el criterio del modelo.

El segundo punto de acuerdo fue la ausencia de fricción operativa. Ninguno dedicó tiempo al sistema salvo el arranque diario, y ambos lo describieron como imperceptible en segundo plano. El tercero fue la adaptabilidad, puntuada con 10 y 9: los dos locales usaron su cámara preexistente, sin cableado nuevo ni equipos adicionales.

La divergencia informativa aparece en la facilidad de uso: P1 otorgó 9 y P2, 7. La diferencia responde al perfil, no al sistema. P2 declaró no manejar plataformas informáticas y necesitó ayuda para ubicar las fotografías. El contraste sugiere que la curva de aprendizaje del panel web depende del alfabetismo digital; el correo funcionó igual para ambos

e. Áreas de Mejora Sugeridas

Cada participante pidió una mejora distinta, y las dos apuntan al mismo lugar: la puerta de entrada al sistema. P1 pidió simplificar la instalación; abrir una carpeta con numerosos archivos le exigió acompañamiento. P2 pidió un panel de control más sencillo, con botones grandes y menos opciones técnicas, para quien no domina la tecnología.

Sobre el canal de notificación, las dos propuestas exceden el alcance de este trabajo. P1 planteó la integración directa con el ECU 911 y observó que las unidades de policía comunitaria cercanas a las zonas comerciales multiplicarían el tiempo que el sistema ahorra. P2 pidió una alerta sonora más fuerte en el teléfono y su envío a un grupo

de mensajería de seguridad barrial, para que se enteren los comercios vecinos. Las dos ideas comparten una lógica: el aviso gana valor cuando alcanza a más de una persona.

Ambos declararon que recomendarían el sistema a otros comerciantes del sector: P1 por su bajo costo de mantenimiento, P2 por la tranquilidad que aporta sin exigir vigilancia manual.

Capítulo IV

Conclusiones y Recomendaciones

Conclusión general:

El hallazgo central cabe en una comparación: la cadena de notificación tardó casi ochenta veces más que el modelo que la origina. En los 100 ciclos ejecutados en dos comercios de Loja, el aviso llegó al receptor en 5.559,8 milisegundos de promedio, con desviación de 854,2 ms y percentil 95 de 6.834 ms. El reparto entre tramos fue muy desigual. La inferencia local consumió 69,8 ms, el 1,3 % del total; el envío al servidor tomó 1.859,1 ms, el 33,4 %; el backend procesó y despachó el correo en 447,3 ms, el 8,0 %; la entrega final al destinatario absorbió 3.183,7 ms, el 57,3 %. El detector, donde se concentró el esfuerzo de desarrollo, casi no pesa en el tiempo que percibe quien recibe el aviso.

De ahí sale una consecuencia práctica. Podar el modelo, cuantizarlo o cambiar de versión devolvería mejoras marginales sobre el tiempo total: el margen real está en el canal de notificación y en el ancho de banda del establecimiento, y ninguna intervención sobre la red neuronal toca esos dos frentes. El modelo, por su parte, sostuvo en operación 82,4 % de precisión, 84,0 % de recall y 83,2 % de F1-score, entre cinco y ocho puntos por debajo del entrenamiento. Es una caída medible que no le quita utilidad operativa al detector. Y el procedimiento quedó documentado: cualquier establecimiento que cumpla las condiciones mínimas puede repetir la medición con el protocolo de instrumentación y ubicar su propio cuello de botella antes de invertir.

Conclusiones según objetivos específicos:

- Diseñar un protocolo de validación replicable que establezca las condiciones mínimas de operación y el procedimiento de instrumentación de marcas de tiempo: El protocolo quedó definido y aplicado. Documenta las condiciones mínimas de operación (hardware de gama media con GPU, red estable, umbral de confianza de 0,75 y cámaras IP sobre RTSP o webcam USB) y el procedimiento de instrumentación, que combina el registro inference_metrics.csv del cliente con las marcas de tiempo del

backend. Con ese material, cualquier establecimiento que cumpla las condiciones repite la medición usando el mismo cliente PyQt5.

- Validar el comportamiento del modelo de detección de armas en espacios públicos comerciales, contrastando su precisión, recall y F1-score en operación real frente a la fase de entrenamiento: El contraste confirmó la estabilidad del modelo fuera del laboratorio. En entrenamiento, el modelo reportó 90,3 % de precisión, 89,4 % de recall, 89,8 % de F1-score y 91,2 % de mAP@0.5. En los 100 ciclos de operación, con 50 escenarios armados y 50 de control, el modelo alcanzó 82,4 % de precisión, 84,0 % de recall y 83,2 % de F1-score: 7,9, 5,4 y 6,6 puntos porcentuales por debajo del laboratorio, respectivamente. La degradación existe y es medible, pero no compromete la utilidad operativa del detector.
- Caracterizar la cadena de notificación en cada uno de sus eslabones: La cadena quedó medida tramo por tramo sobre las 51 alertas registradas en los 100 ciclos. T1, de la captura del cuadro a la detección positiva con evidencia guardada, promedió 69,8 ms; T2, el envío al servidor, 1.859,1 ms; T3, el procesamiento del backend y el despacho del correo por la API de Brevo, 447,3 ms; y T4, la entrega al buzón del destinatario, 3.183,7 ms. Este último tramo se midió a partir de las cabeceras Received del correo, de modo que incorpora tanto la latencia de la infraestructura del proveedor como la del cliente de correo del receptor.
- Identificar los puntos de retraso recurrentes en la cadena de notificación y formular recomendaciones técnicas y operacionales: El retraso apareció lejos del modelo. Dos tramos cargan con el 90,7 % del tiempo: la entrega del correo al destinatario, con el 57,3 %, y el envío de la imagen al servidor, con el 33,4 %. El backend aporta el 8,0 % y la inferencia local el 1,3 % restante. Ese diagnóstico alimentó las recomendaciones: comprimir las imágenes, mantener un pool de conexiones HTTP, enviar de forma asíncrona y evaluar canales con garantías de entrega más estrictas, junto con las operacionales sobre calidad de red y protocolo de respuesta del establecimiento.

Recomendaciones

- Para futuras investigaciones académicas: el dataset es el frente más claro. Sumarle imágenes bajo lluvia, neblina y poca luz, con más ángulos y resoluciones, ataca de frente la debilidad que el Capítulo I identificó en los detectores revisados: se degradan fuera de sus condiciones de entrenamiento. También queda abierto comparar, sobre este mismo dataset, arquitecturas posteriores a YOLOv8 como YOLO-NAS o EfficientDet, para separar cuánto del rendimiento pone el modelo y cuánto ponen los datos.
- Los diecisiete errores registrados apuntan a dos frentes. El primero no es del modelo: los ocho falsos negativos vinieron de oclusión por el ángulo de cámara. Las cámaras operaron a dos metros y medio del suelo, la altura que recomienda el Manual de Usuario (Anexo E, §11), y aun así la franja del mostrador quedó fuera de vista; corregirlo exige revisar la inclinación y el punto de montaje en cada local. El segundo sí es del dataset. Un mismo taladro negro de mango curvo provocó cinco de los nueve falsos positivos, y una billetera oscura abierta, otros dos. Entrenar con imágenes negativas de herramientas y objetos oscuros de mango recortaría, por sí solo, más de la mitad de las falsas alarmas observadas.
- Para desarrolladores e implementadores: valdría la pena jerarquizar las alertas de baja a alta prioridad y agregar un módulo de listas blancas, pensado para los contextos donde circulan portadores autorizados de armas, como militares o guardias de seguridad privada, que hoy dispararían falsos positivos.
- Para entidades públicas y privadas interesadas en adoptar la solución: nada sustituye al piloto. En los dos comercios de Loja, cada sitio impuso condiciones propias de iluminación, posición de cámaras y calidad de red que ningún despliegue masivo anticipa desde el escritorio. Antes de escalar conviene resolver tres cosas en pequeño. Integrar el sistema con las cámaras y la vigilancia ya instaladas. Definir quién recibe el correo y qué hace en los segundos siguientes, es decir, dónde encaja

la alerta en el protocolo de respuesta del local. Y capacitar al personal en el dashboard, sobre todo en distinguir una detección válida de un falso positivo.

- En cuanto al rendimiento del sistema: conviene operar sobre equipos con GPU, comprimir las imágenes antes de subirlas y asegurar una conexión estable. Los números del estudio respaldan las tres cosas: T1 se mantuvo en 69,8 ms gracias a la GPU, mientras que T2, el tramo que depende de la imagen y de la red, promedió 1.859,1 ms con picos de 3.419 ms.
- Respecto a la validación con usuarios: dos participantes alcanzan para una lectura inicial, no para conclusiones de usabilidad. Repetir la entrevista del Anexo H con más comerciantes y en horarios de alto flujo afinaría la interfaz y las expectativas de tiempo de respuesta frente a las necesidades reales del entorno.

Referencias

Amazon Web Services. (2024). *¿Qué es la visión artificial?* Obtenido de AWS: https://aws.amazon.com/es/what-is/computer-vision/

Arana, I. (18 de Mayo de 2019). *China por el reconocimiento facial*. Obtenido de La Vanguardia: https://www.lavanguardia.com/tecnologia/20190518/462270404745/reconocimiento-facial-china-derechos-humanos.html

Banco Mundial. (2024). *Homicidios intencionales (por cada 100.000 habitantes) – Ecuador.* Obtenido de Datos del Banco Mundial.

Brodowicz, M. (16 de Junio de 2024). *El impacto del aprendizaje profundo en la inteligencia artificial y sus aplicaciones en la vida cotidiana*. Obtenido de Aithor: https://aithor.com/essay-examples/el-impacto-del-aprendizaje-profundo-en-la-inteligencia-artificial-y-sus-aplicaciones-en-la-vida-cotidiana

De la Cruz, M. (20 de Febrero de 2024). *Análisis bibliométrico de la aplicación de redes neuronales convolucionales en el área de la visión artificial*. Obtenido de Universidad Nacional Hermilio Valdizán: https://www.redalyc.org/journal/5860/586077946002/html/

Forruque, S. (17 de Abril de 2023). *Técnicas de modelado de aprendizaje profundo: avances actuales, aplicaciones, ventajas y desafíos*. Obtenido de Springer: https://link.springer.com/article/10.1007/s10462-023-10466-8

Fraile, M. (1 de Julio de 2024). *Metadesigners: Symbiosis of Digital Tools and Artificial Intelligence in Architectural Design*. Obtenido de Universidad Rey Juan Carlos Madrid, España: https://revistas.uasb.edu.ec/index.php/uru/article/view/4865/5020

Google Cloud. (2024). *¿Qué es el aprendizaje automático (AA)?* Obtenido de https://cloud.google.com/learn/what-is-machine-learning?hl=es-419

Gutiérrez Velázquez, M. Á. (5 de Julio de 2023). *A Survey on Problems of Detection of Objects in Digital Images and Videos*. Obtenido de Universidad de Guadalajara: https://www.redalyc.org/journal/5122/512275598008/html/

Gutiérrez, D. (2021). *Diseño y desarrollo de un sistema de video vigilancia basado en dispositivos embebidos, tecnicas de vision artificial y algoritmos inteligentes*. Obtenido de Universidad Politécnica Salesiana: https://dspace.ups.edu.ec/bitstream/123456789/19956/1/UPS-CT008985.pdf

Hernández, C. (Marzo de 2025). *Sistemas de vigilancia basados en inteligencia artificial en la prevención del delito*. Obtenido de ResearchGate: https://www.researchgate.net/publication/390100946_Sistemas_de_vigilancia_basados_en_inteligencia_artificial_en_la_prevencion_del_delito_en_Peru

Hong, Z. (Junio de 2021). *Multi-Scale Ship Detection from SAR and Optical Imagery via A More Accurate YOLO*. Obtenido de ResearchGate: https://www.researchgate.net/figure/A-comparison-of-the-Faster-R-CNN-SSD-CenterNet-and-YOLOv3-network-frameworks_fig2_352279048

Instituto Nacional de Estadística y Censos. (2024). *Ecuador en cifras*. Obtenido de INEC: https://www.ecuadorencifras.gob.ec/justicia-y-crimen/

International Business Machines. (9 de Agosto de 2024). *¿Qué es la inteligencia artificial (IA)?* Obtenido de IBM: https://www.ibm.com/mx-es/think/topics/artificial-intelligence

IONOS Digital Guide. (2024). *Convolutional Neural Network: ¿qué es una red neuronal convolucional (CNN)?* Obtenido de Ionos: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/convolutional-neural-network/

Jayaswal, V. (14 de Septiembre de 2020). *Métricas de rendimiento: matriz de confusión, precisión, recuperación y puntuación F1*. Obtenido de TOWARDS: https://towardsdatascience.com/performance-metrics-confusion-matrix-precision-recall-and-f1-score-a8fe076a2262/

Jiménez, L. (15 de Marzo de 2025). *Guía Completa de Detección de Objetos: De la Teoría a la Implementación*. Obtenido de Linkedin: https://es.linkedin.com/pulse/gu%C3%ADa-completa-de-detecci%C3%B3n-objetos-la-teor%C3%ADa-jimenez-alvarez-lyswe

Ley Orgánica de Protección de Datos Personales. (2021). *Registro Oficial Quinto Suplemento N.º 459.* Obtenido de LOPDP: https://www.finanzaspopulares.gob.ec/wp-content/uploads/2021/07/ley_organica_de_proteccion_de_datos_personales.pdf

Librería CATEDU. (2023). *Inicios y evolución de la IA*. Obtenido de CATEDU: https://libros.catedu.es/books/la-ia-en-educacion-una-aproximacion-practica/page/unidad-13-inicios-y-evolucion-de-la-ia/revisions/46220/changes

Miao, F. (2021). *Inteligencia artificial y educación: guía para las personas a cargo de formular políticas*. Obtenido de UNESCO: https://unesdoc.unesco.org/ark:/48223/pf0000379376

Mirmehdi, M. (2020). *Visión por computadora*. Obtenido de ScienceDirect: https://www.sciencedirect.com/topics/engineering/computervision

Nvidia. (2025). *Inferencia de Cloud a Edge para Ciudades con IA Más Inteligentes*. Obtenido de NVIDIA: https://www.nvidia.com/es-la/deep-learning-ai/inference-platform/intelligent-video-analytics/

Observatorio Ecuatoriano de Crimen Organizado. (2024). *Boletín anual de homicidios intencionales en Ecuador: Variación de los homicidios intencionales en Ecuador (Boletín anual N.º 1, enero–diciembre 2023).* Obtenido de Fundación Panamericana para el Desarrollo: https://oeco.padf.org/wp-content/uploads/2024/04/OECO.-BOLETIN-ANUAL-DE-HOMICIDIOS-2023.pdf

Observatorio Ecuatoriano de Crimen Organizado. (2024). *Radiografía de la extorsión: Tipologías y resultado de la Encuesta Nacional de Victimización de Casos de*

Extorsión Empresarial. Obtenido de Fundación Panamericana para el Desarrollo: https://oeco.padf.org/wp-content/uploads/2024/05/OECO_Radiografia-de-la-extorsion-en-Ecuador-1.pdf

Pedraza, J. D. (Mayo de 2023). *La Inteligencia Artificial en la sociedad: Explorando su Impacto Actual y los Desafíos Futuros*. Obtenido de Universidad Politécnica de Madrid: https://oa.upm.es/75068/1/TFG_JAROD_DAVID_PEDRAZA_CARO.pdf

Pérez, A. (12 de Febrero de 2021). *Challenges of automated video surveillance*. Obtenido de Dialnet: https://dialnet.unirioja.es/descarga/articulo/9048925.pdf

Reyes, O. J. (2020). *Técnicas de inteligencia artificial utilizadas en el procesamiento de imágenes y su aplicación en el análisis de pavimentos*. Obtenido de Escuela de Ingeniería de Antioquía: https://www.redalyc.org/journal/1492/149258931014/html/

Sánchez, L. A. (10 de Junio de 2022). *Inteligencia artificial y visión por computadora aplicada a la educación*. Obtenido de Jalasoft, Ecuador: https://revista.uisrael.edu.ec/index.php/ro/article/view/587/631

Ultralytics. (Mayo de 2024). *YOLOv5 vs YOLOv8: comparación detallada*. Obtenido de https://docs.ultralytics.com/es/compare/yolov5-vs-yolov8/

Unapucha, J. (2025). *Sistema CCTV con alerta temprana y reconocimiento de armas de fuego utilizando Machine Learning*. Obtenido de Universidad Técnica de Ambato: https://repositorio.uta.edu.ec/server/api/core/bitstreams/394c7de0-6df7-42d5-a34a-61bee8f23cae/content

Villarreal, M. E. (28 de Diciembre de 2023). *Desafíos de los sistemas de armas autónomas y formas de abordarlos: una perspectiva desde la academia y la sociedad civil latinoamericana*. Obtenido de UNODA: https://docs-library.unoda.org/General_Assembly_First_Committee_-Seventy-Ninth_session_(2024)/78-241-Latin_America-SP.pdf

Virtudes, J. (2023). *Reconocimiento de objetos con Deep Learning*. Obtenido de DAMAVIS: https://blog.damavis.com/reconocimiento-de-objetos-con-deep-learning/