



**UNIVERSIDAD MARIANO GÁLVEZ DE
GUATEMALA**

**INGENIERÍA EN SISTEMAS DE INFORMACIÓN
Y CIENCIAS DE LA COMPUTACIÓN**

ARQUITECTURA DE COMPUTADORAS II

INGE.ALEX GARCIA

Clúster de Edge Computing para inferencia de IA local autónoma

WILMER ANTONIO COLINDRES CHOC
ANGEL ENMANUEL VÁSQUEZ MAZARIEGOS
CRISTIAN ALEXANDER GONZÁLEZ CASTILLO
ARNOLDO IVÁN GONZÁLEZ PÉREZ

0900-17-990
0900-23-14533
0900-23-18231
0900-12-10245

GUATEMALA 16 DE OCTUBRE 2026



Resumen ejecutivo

EdgeVision Inclusivo es un prototipo funcional de Edge Computing que ejecuta inferencia de visión artificial dentro de una red local, sin depender de servicios públicos de nube. Una Raspberry Pi 4B captura video, coordina dos servicios remotos y presenta los resultados en un panel web; el primer worker procesa detección de objetos con YOLO26n en formato NCNN; el segundo clasifica seis señas de apoyo AGUA, AYUDA, BAÑO, EMERGENCIA, NO y SÍ. Mediante un modelo personalizado también exportado a NCNN. La inferencia se distribuye por HTTP y las solicitudes se realizan en paralelo para evitar que las latencias de ambos workers se acumulen de forma secuencial.

La solución incorpora tolerancia a fallos a nivel de aplicación, cuando un worker no responde, el nodo maestro ejecuta el modelo correspondiente de manera local y conserva el servicio, aunque con menor rendimiento. En las pruebas observadas, la operación distribuida alcanzó aproximadamente 15 a 16 FPS con latencias de 48 a 58 ms mostradas por el panel; una ejecución local previa registró 2.3 FPS y 417 ms. Durante la caída del worker de señas, el panel mantuvo la inferencia mediante el fallback del maestro y posteriormente recuperó el procesamiento distribuido cuando el nodo regresó. Estas cifras son mediciones operativas del prototipo y no sustituyen una evaluación estadística de exactitud del modelo.

El clúster K3s se configuró con tres nodos y se validó mediante un despliegue Nginx de dos réplicas distribuidas entre ambos workers. Las APIs de inferencia, por su parte, se ejecutan actualmente como servicios systemd en las máquinas worker.

Contenido

1. Introducción	1
2. Planteamiento del problema	1
3. Objetivos.....	1
3.1 Objetivo general	1
3.2 Objetivos específicos.....	2
4. Alcance y restricciones	2
5. Fundamento técnico	2
5.1 Edge Computing.....	2
5.2 K3s y Kubernetes	3
5.3 YOLO y NCNN	3
6. Arquitectura de la solución	3
7. Componentes	4
7.1 Hardware.....	4
7.2 Software	4
8. Red y servicios	5
9. Modelos y datos.....	5
9.1 Detección de objetos	5
9.2 Clasificación de señas.....	5
10. Implementación	6
11. Distribución y tolerancia a fallos.....	7
12. Pruebas y resultados	8
12.1 Casos ejecutados.....	8
12.2 Comparación operativa	9
12.3 Evidencia visual.....	9

13. Seguridad y operación local	12
14. Limitaciones y mejoras.....	12
15. Conclusiones	13
16. Glosario	14
17. Referencias.....	15

1. Introducción

La computación en el borde acerca el procesamiento a la fuente de los datos. En una aplicación de video, esta decisión reduce la dependencia de Internet, evita transferir continuamente imágenes a servicios externos y permite mantener tiempos de respuesta apropiados para interacción humana. El proyecto aplica este principio a un escenario de apoyo inclusivo: reconocer objetos comunes y un conjunto acotado de señas desde una cámara conectada al nodo coordinador.

La propuesta se desarrolló como un clúster heterogéneo, la Raspberry Pi representa un equipo de borde de recursos limitados y los workers aportan capacidad de CPU adicional mediante máquinas virtuales. Esta combinación permite demostrar coordinación distribuida, comunicación por red privada, despliegue de servicios persistentes, observabilidad básica y continuidad de operación ante una falla controlada.

2. Planteamiento del problema

La ejecución simultánea de detección de objetos y clasificación de señas en una sola Raspberry Pi aumenta la carga de CPU y eleva la latencia por cuadro. Además, una arquitectura concentrada convierte al proceso local en el único punto de ejecución de los modelos. El reto consiste en repartir la carga entre nodos accesibles en una red local y conservar la inferencia cuando uno de ellos queda temporalmente fuera de servicio.

La solución debía operar sin una nube pública y utilizar modelos ligeros compatibles con el entorno, también debía permitir demostrar el estado del clúster y la ubicación del trabajo. Por ello se combinaron servicios HTTP de inferencia, modelos NCNN, un panel Flask y K3s como evidencia de orquestación liviana.

3. Objetivos

3.1 Objetivo general

Diseñar, implementar y validar un clúster de Edge Computing capaz de procesar localmente video para detección de objetos y clasificación de señas, distribuyendo la inferencia entre tres nodos y manteniendo continuidad del servicio ante la indisponibilidad de un worker.

3.2 Objetivos específicos

- Configurar una red privada con direcciones estáticas para el nodo maestro y los dos workers.
- Ejecutar modelos de IA optimizados en formato NCNN para detección y clasificación.
- Exponer la inferencia remota mediante APIs HTTP y consumirlas concurrentemente desde el maestro.
- Visualizar video, resultados, latencia, FPS, origen de inferencia y estado de los workers.
- Implementar fallback local por modelo y comprobar la recuperación después de una falla.
- Validar un clúster K3s de tres nodos con un despliegue distribuido entre los workers.

4. Alcance y restricciones

Elemento	Decisión de alcance
Incluido	Raspberry Pi como maestro, cámara, dos workers virtuales, red local, APIs de inferencia, panel web, K3s y prueba de fallo.
Inferencia	Detección de objetos con YOLO26n NCNN y clasificación de seis señas con un modelo NCNN personalizado.
Límite actual	Las APIs de IA se administran con systemd; K3s demuestra el clúster y la distribución de un despliegue de prueba.

5. Fundamento técnico

5.1 Edge Computing

Edge Computing desplaza el cómputo desde centros remotos hacia nodos cercanos a la captura, en este proyecto la cámara, el coordinador y los workers se encuentran en la misma subred. El flujo puede continuar sin acceso a Internet después de instalar los componentes, y las imágenes se procesan dentro de la infraestructura local.

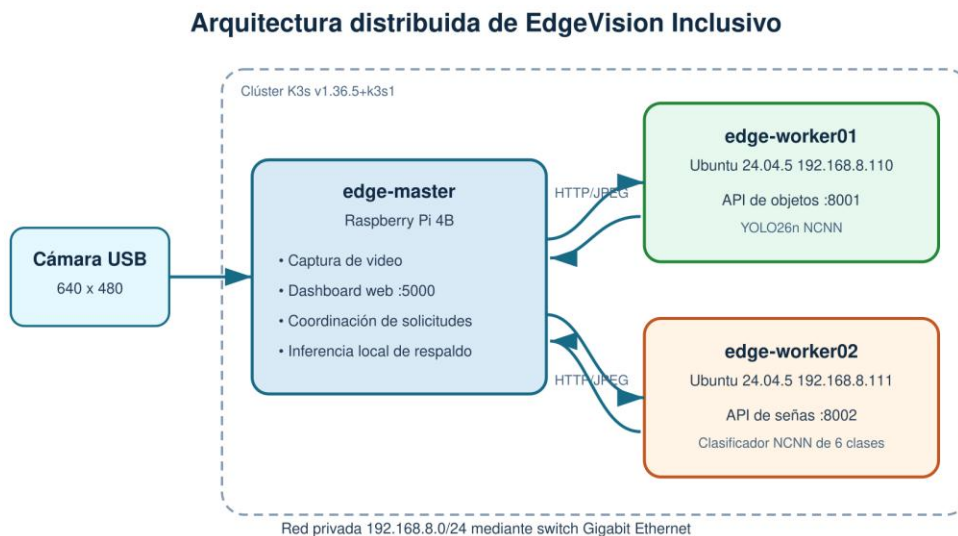
5.2 K3s y Kubernetes

K3s es una distribución ligera de Kubernetes orientada a recursos limitados, entornos de borde y arquitecturas ARM, el servidor instalado en la Raspberry Pi mantiene el plano de control y los dos workers se registran como agentes. Kubernetes aporta conceptos de nodos, réplicas, programación de cargas y recuperación declarativa. La prueba Nginx con anti-afinidad evidencia que las réplicas pueden ubicarse en nodos distintos.

5.3 YOLO y NCNN

Ultralytics permite ejecutar tareas de detección y clasificación y exportar modelos al formato NCNN, este está diseñado para inferencia eficiente en dispositivos y sistemas con recursos restringidos, ambos modelos operativos se distribuyeron como directorios con archivos de parámetros, pesos binarios y metadatos, lo que evita cargar modelos de entrenamiento completos durante la ejecución.

6. Arquitectura de la solución



El nodo maestro obtiene cada cuadro mediante OpenCV. A partir del mismo JPEG genera dos solicitudes concurrentes: una hacia el puerto 8001 del worker de objetos y otra hacia el puerto 8002 del worker de señas, cada servicio protege el acceso al modelo con un bloqueo de exclusión mutua, decodifica la imagen, realiza la predicción y devuelve JSON. El maestro integra ambas respuestas, dibuja los resultados y publica el panel en el puerto 5000.

K3s utiliza el puerto 6443 para el API del clúster, la validación de orquestación se realizó con Nginx expuesto mediante NodePort 30080 y dos réplicas, una en cada worker. Las APIs de inferencia permanecen fuera de los pods y se inician automáticamente con systemd. Esta arquitectura híbrida permitió concluir el prototipo funcional y conserva una ruta clara de migración posterior hacia Deployment y Service de Kubernetes.

7. Componentes

7.1 Hardware

Componente	Uso
Raspberry Pi 4B, 4 GB	Captura de cámara, coordinación, dashboard, K3s server y fallback local.
Cámara USB/V4L2	Fuente de video para ambos modelos.
Worker 01 virtual AMD64	Servicio remoto de detección de objetos.
Worker 02 virtual AMD64	Servicio remoto de clasificación de señas.
Red local Gigabit	Transporte de imágenes JPEG, respuestas JSON y tráfico del clúster.

7.2 Software

Capa	Tecnología / versión
Maestro	Debian GNU/Linux 13, ARM64, Python 3.13.5.
Workers	Ubuntu 24.04.5 LTS, AMD64, Python 3.12.3.
IA	Ultralytics 8.4.174, NCNN 1.0.20260526.
Video/API	OpenCV 5.0.0.93, NumPy 2.5.2, Flask 3.1.3.
Clúster	K3s v1.36.5+k3s1, containerd 2.3.4-k3s1.36.
Persistencia	Servicios systemd: edgevision, edgevision-object y edgevision-sign.

8. Red y servicios

Nodo	IP	Sistema	Servicio principal	Puerto
edge-master	192.168.8.100	Debian 13 ARM64	Dashboard / K3s API	5000 / 6443
edge-worker01	192.168.8.110	Ubuntu 24.04 AMD64	Objetos / K3s agent	8001
edge-worker02	192.168.8.111	Ubuntu 24.04 AMD64	Señas / K3s agent	8002
Servicio de prueba	Red local	K3s	Nginx NodePort	30080

La separación por puertos facilita verificar cada componente con rutas de salud, los endpoints /health devuelven el nombre del nodo, el tipo de servicio y el estado. Los endpoints /infer aceptan una imagen multipart/form-data y responden con detecciones o con la clase, confianza y latencia. El panel consulta de forma indirecta estos resultados al procesar el flujo de cámara.

9. Modelos y datos

9.1 Detección de objetos

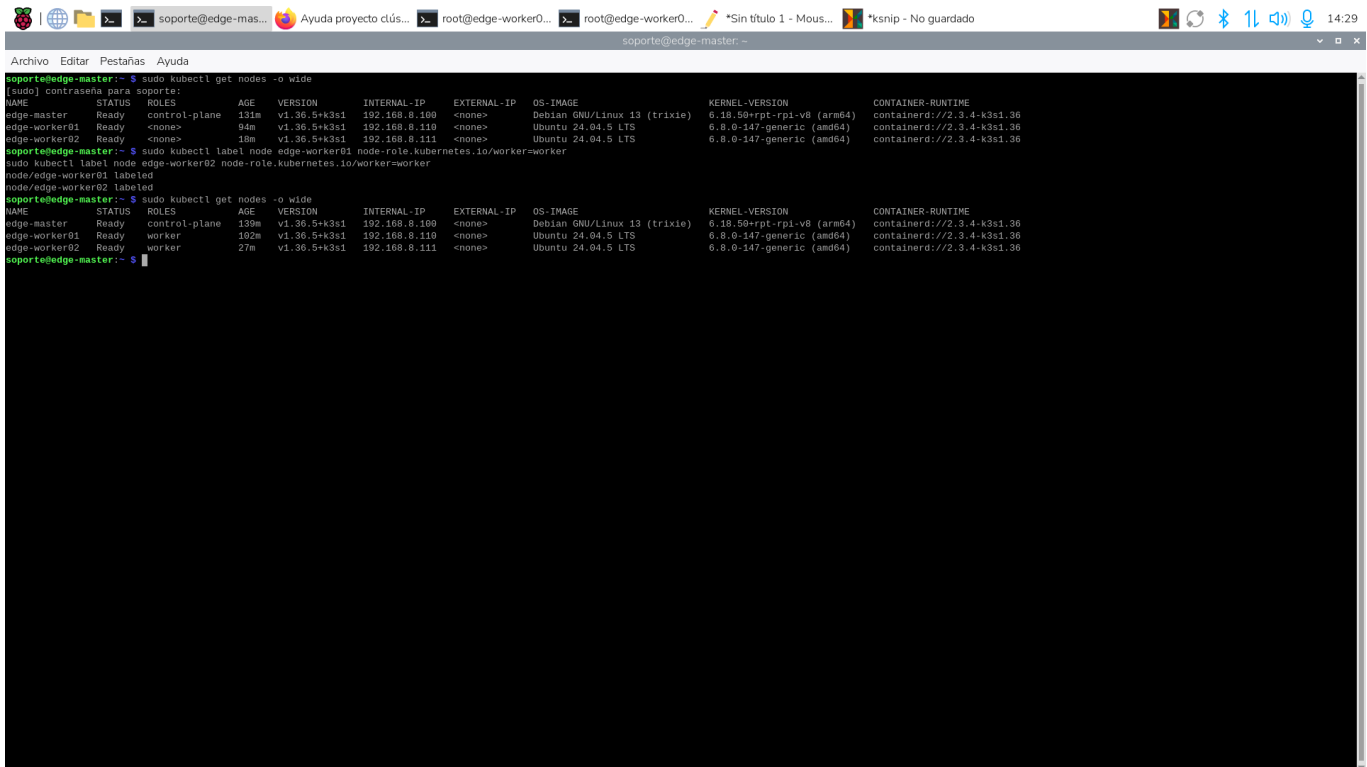
El modelo de objetos es YOLO26n exportado a NCNN. Recibe el cuadro completo y produce cajas delimitadoras, identificadores de clase y confianzas. El dashboard traduce al español un conjunto de clases comunes, como persona, botella, silla, computadora portátil y teléfono. La API del worker devuelve las coordenadas de cada caja para que el maestro las represente.

9.2 Clasificación de señas

El clasificador personalizado reconoce seis clases: agua, ayuda, baño, emergencia, no y sí. Para reducir cambios inestables, el maestro mantiene un historial corto y solo acepta una seña cuando alcanza el umbral de confianza y la estabilidad configurada. El modelo clasifica el cuadro completo; por ello, el encuadre, la iluminación y la presencia de varios sujetos afectan el resultado.

10. Implementación

1. Se preparó la Raspberry Pi con cámara, entorno virtual, modelos NCNN y aplicación Flask.
2. Se configuraron direcciones IP estáticas y conectividad entre los tres nodos.
3. Se instaló K3s server en el maestro y se unieron los dos agentes mediante token.
4. Se desplegó Nginx con dos réplicas y anti-afinidad para validar distribución en los workers.
5. Se copiaron el modelo de objetos al worker01 y el modelo de señas al worker02.
6. Se crearon entornos Python CPU-only y APIs Flask independientes en los workers.
7. Se registraron los servicios edgevision-object y edgevision-sign en systemd.
8. Se adaptó el dashboard para despachar las dos solicitudes en paralelo y activar fallback local.
9. Se ejecutaron pruebas de salud, inferencia, caída y recuperación de los workers.



The screenshot shows a terminal window with the following content:

```
soporte@edge-master:~$ sudo kubectl get nodes -o wide
[sudo] contraseña para soporte:
NAME          STATUS    ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION      CONTAINER-RUNTIME
edge-master   Ready     control-plane  131m   v1.36.5+k3s1  192.168.8.100   <none>         Debian GNU/Linux 13 (trixie)  6.10.50+rpt-rpi-v8 (arm64)  containerd://2.3.4-k3s1.36
edge-worker01 Ready     <none>     94m    v1.36.5+k3s1  192.168.8.110   <none>         Ubuntu 24.04.5 LTS      6.8.0-147-generic (amd64)  containerd://2.3.4-k3s1.36
edge-worker02 Ready     <none>     18m    v1.36.5+k3s1  192.168.8.111   <none>         Ubuntu 24.04.5 LTS      6.8.0-147-generic (amd64)  containerd://2.3.4-k3s1.36
soporte@edge-master:~$ sudo kubectl label node edge-worker01 node-role.kubernetes.io/worker=worker
node/edge-worker01 labeled
soporte@edge-master:~$ sudo kubectl label node edge-worker02 node-role.kubernetes.io/worker=worker
node/edge-worker02 labeled
soporte@edge-master:~$ sudo kubectl get nodes -o wide
NAME          STATUS    ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION      CONTAINER-RUNTIME
edge-master   Ready     control-plane  139m   v1.36.5+k3s1  192.168.8.100   <none>         Debian GNU/Linux 13 (trixie)  6.10.50+rpt-rpi-v8 (arm64)  containerd://2.3.4-k3s1.36
edge-worker01 Ready     worker    102m   v1.36.5+k3s1  192.168.8.110   <none>         Ubuntu 24.04.5 LTS      6.8.0-147-generic (amd64)  containerd://2.3.4-k3s1.36
edge-worker02 Ready     worker    27m    v1.36.5+k3s1  192.168.8.111   <none>         Ubuntu 24.04.5 LTS      6.8.0-147-generic (amd64)  containerd://2.3.4-k3s1.36
soporte@edge-master:~$
```

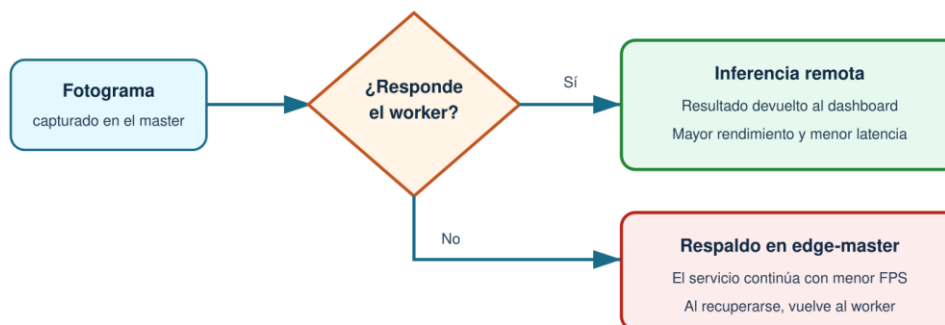
```
soporte@edge-master:~$ sudo kubectl get deployment,service,pods -o wide
NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS IMAGES SELECTOR
deployment.apps/prueba-cluster 2/2 2 2 6m18s web nginx:alpine app=prueba-cluster

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE SELECTOR
service/kubernetes ClusterIP 10.43.0.1 <none> 443/TCP 147m <none>
service/prueba-cluster NodePort 10.43.20.201 <none> 80:30880/TCP 6m18s app=prueba-cluster

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
pod/prueba-cluster-5944bf8cc-pcgkt 1/1 Running 0 6m18s 10.42.2.3 edge-worker02 <none> <none>
pod/prueba-cluster-5944bf8cc-w6svj 1/1 Running 0 6m18s 10.42.1.3 edge-worker01 <none> <none>
soporte@edge-master:~$
```

11. Distribución y tolerancia a fallos

Mecanismo de continuidad de inferencia



El maestro lanza las llamadas a objetos y señas de forma concurrente, si ambas finalizan correctamente, cada resultado se identifica con el hostname remoto. Si una llamada supera el tiempo límite, produce una excepción o devuelve una respuesta inválida, únicamente ese modelo se procesa en el maestro. El otro worker puede continuar activo, esta granularidad evita que la falla de un servicio fuerce el regreso completo a ejecución local.

El panel muestra el origen de cada resultado, el estado de los workers y contadores de fallback, la recuperación es automática porque el siguiente cuadro vuelve a intentar el endpoint remoto. Cuando el servicio responde, la fuente cambia nuevamente al worker, el mecanismo prioriza continuidad; no equivale a reprogramación de pods porque las APIs todavía se ejecutan como unidades systemd.

12. Pruebas y resultados

12.1 Casos ejecutados

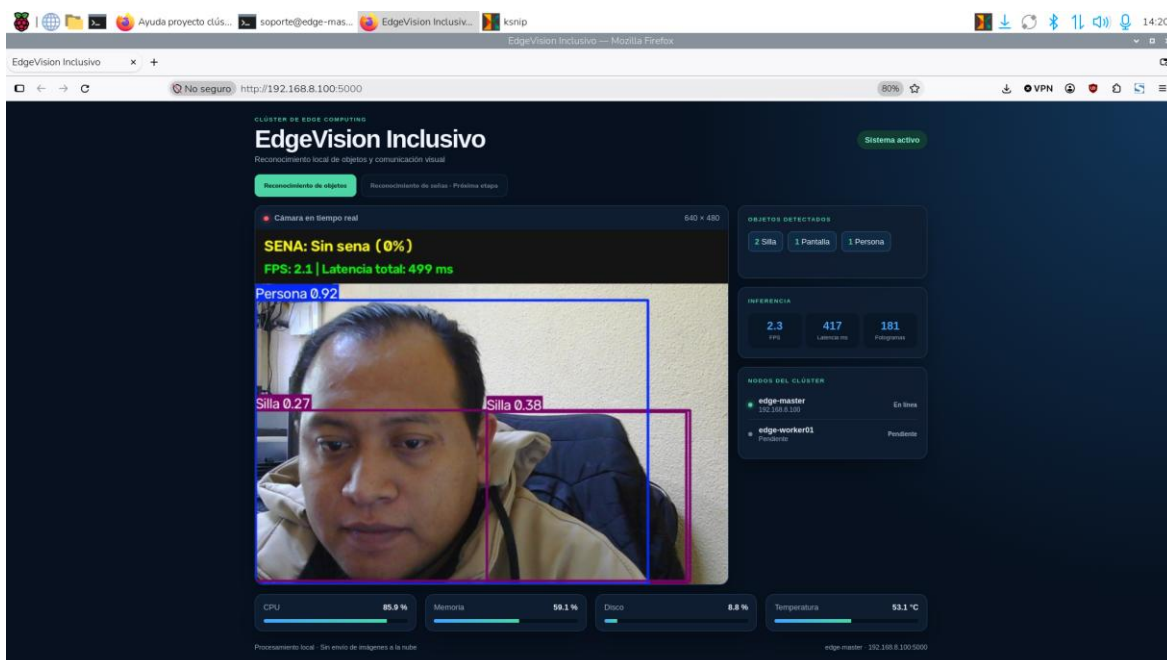
Prueba	Procedimiento	Resultado
Salud worker01	GET http://192.168.8.110:8001/health	Servicio object-detection en estado ok.
Inferencia worker01	POST de bus.jpg al endpoint /infer	Bus y tres personas; latencia API 2088.73 ms en prueba aislada.
Salud worker02	GET al puerto 8002	Servicio sign-classification disponible.
Operación distribuida	Cámara activa con ambas APIs	Aproximadamente 15–16 FPS y 48–58 ms mostrados por el dashboard.
Falla de señas	Worker02 fuera de línea	Continuidad por fallback local; contador observado en 314.
Recuperación	Restablecimiento del servicio worker02	El panel volvió a indicar ambas fuentes remotas y estado online.

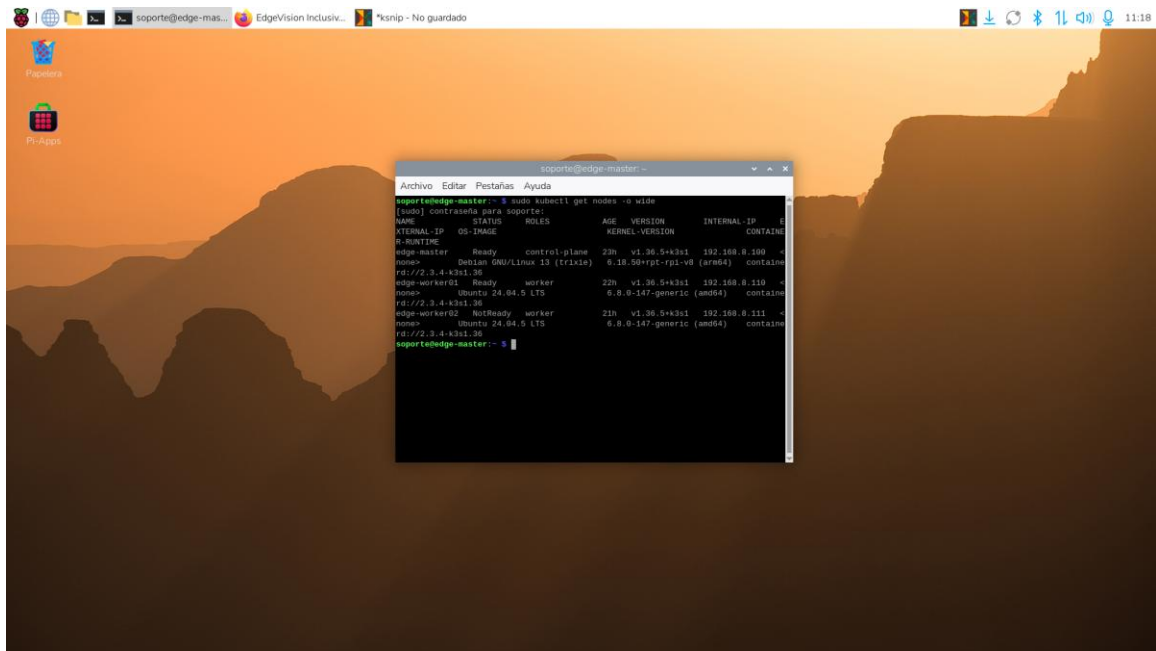
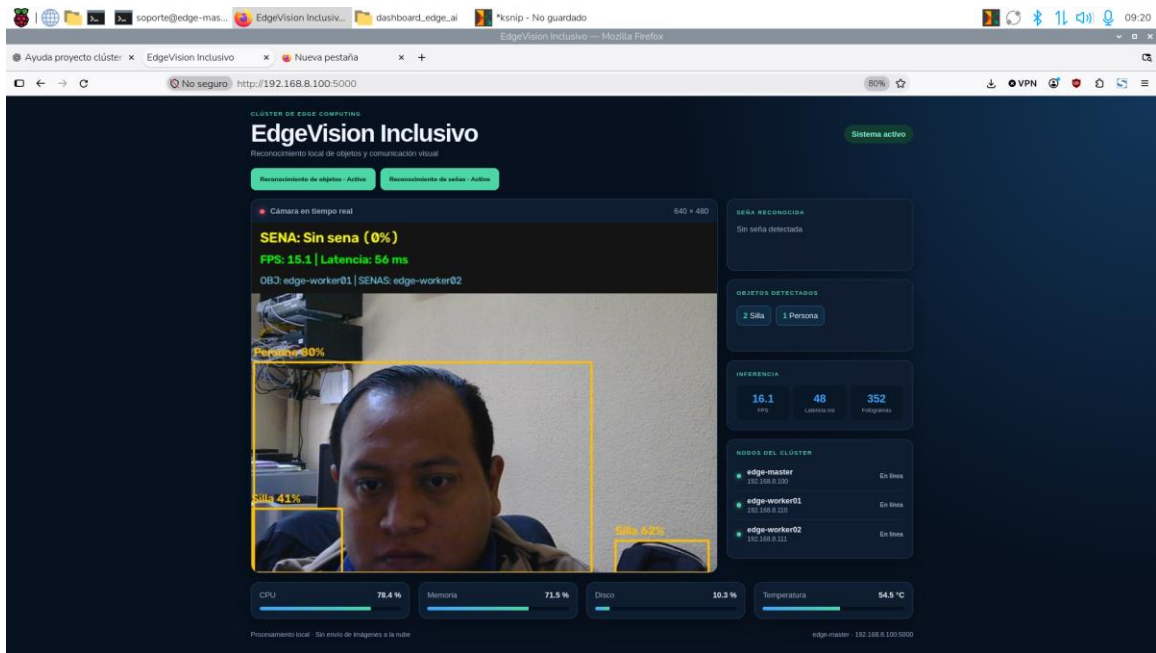
12.2 Comparación operativa

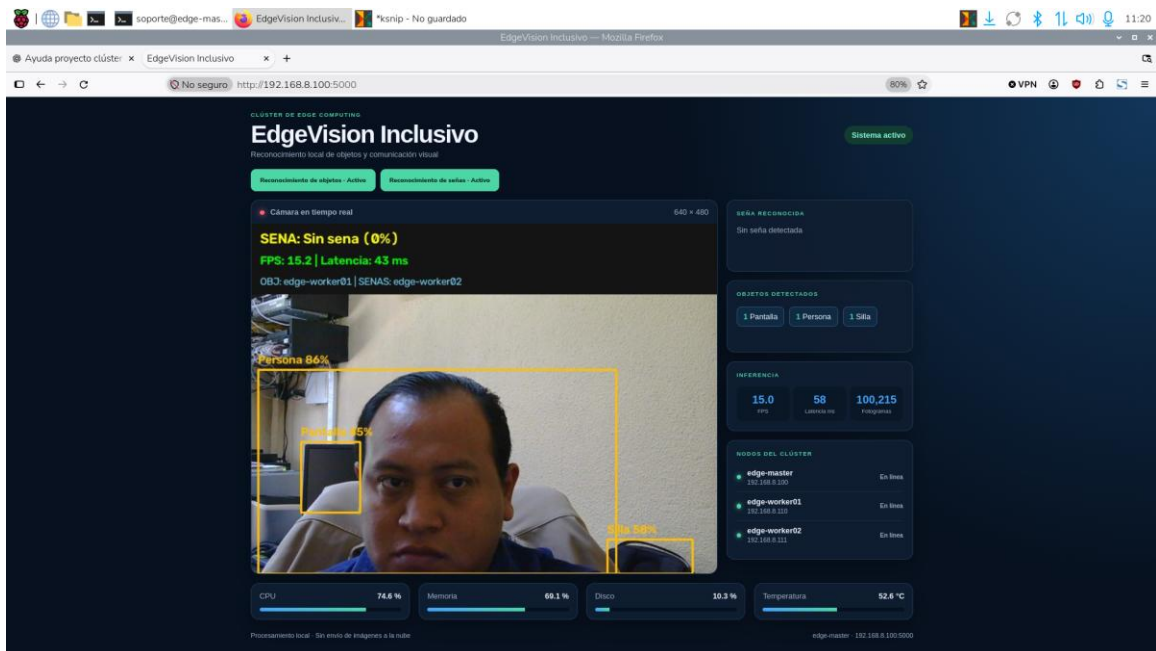
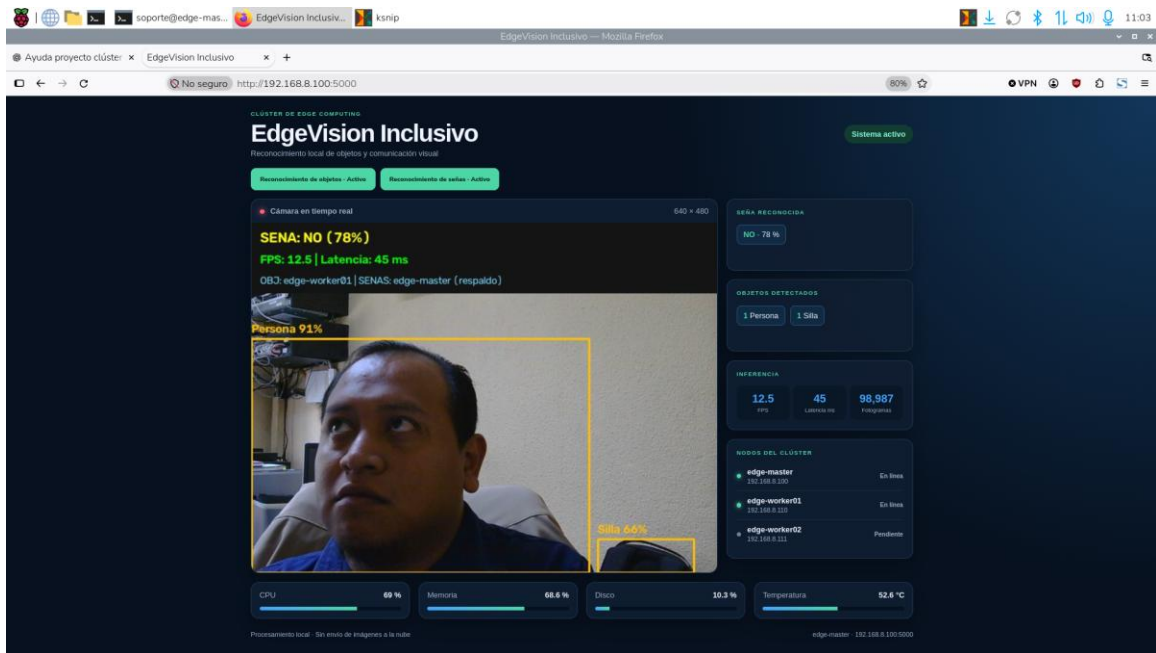
Escenario observado	FPS	Latencia mostrada	Interpretación
Ejecución local inicial	2.3	417 ms	Carga concentrada en el maestro.
Local optimizado / muestra	4.8	141 ms	Seña SÍ al 88 %; workers pendientes.
Distribuida	15.1–16.1	48–56 ms	Objetos en worker01 y señas en worker02.
Distribuida estable	≈15.0	58 ms	Ambos workers online.
Fallback de señas	12.5	45 ms*	Worker02 offline; señas en el maestro.

La prueba distribuida muestra una mejora visible de fluidez frente a la captura local de 2.3 FPS, sin embargo las mediciones provienen de capturas del panel bajo condiciones distintas. Para una comparación rigurosa se requiere repetir cada escenario con el mismo video, registrar percentiles de latencia, uso de CPU, cuadros perdidos y temperatura, y calcular promedios con intervalos definidos.

12.3 Evidencia visual







13. Seguridad y operación local

El sistema mantiene los cuadros dentro de la red privada y no utiliza APIs de terceros para inferencia, esta propiedad reduce exposición de imágenes y permite operar sin Internet. Los servicios están delimitados por direcciones y puertos conocidos, y systemd permite reinicio automático después del arranque o una terminación inesperada.

Para una instalación permanente se recomienda restringir los puertos mediante firewall al segmento autorizado, ejecutar cada servicio con un usuario dedicado sin privilegios, proteger los endpoints con autenticación, limitar el tamaño de carga, rotar registros y almacenar secretos de K3s con controles apropiados. La red de laboratorio demuestra funcionalidad, pero no debe asumirse endurecida para exposición pública.

14. Limitaciones y mejoras

- Contenerizar `object_api` y `sign_api` y administrarlas como Deployment y Service de K3s.
- Versionar el conjunto de datos con manifiesto, particiones fijas, hashes y licencia de uso.
- Calcular exactitud, precisión, recall, F1 y matriz de confusión en un conjunto independiente.
- Ampliar variaciones de iluminación, distancia, fondo, lateralidad y participantes.
- Registrar latencia p50/p95, FPS sostenidos, uso de CPU/RAM, tráfico y tiempo de recuperación.
- Agregar backoff o circuit breaker para reducir intentos durante una indisponibilidad prolongada.
- Incorporar almacenamiento de métricas y alertas sin comprometer la operación local.
- Evaluar workers físicos de bajo consumo como extensión, sin invalidar las VMs .

15. Conclusiones

El prototipo cumple el objetivo de ejecutar inferencia local distribuida en tres nodos, la Raspberry Pi conserva el control de la cámara y la interfaz, mientras los workers procesan tareas especializadas. La concurrencia evita sumar de manera directa los tiempos de ambas inferencias y las capturas muestran una mejora sustancial de fluidez respecto de la ejecución local inicial.

La prueba de fallo demuestra continuidad: al quedar worker02 fuera de línea, el clasificador se ejecutó en el maestro; al restablecerlo, el origen regresó al worker remoto. La solución no oculta la degradación, pues presenta el estado y los contadores. Esto cumple el propósito funcional de tolerancia a fallos a nivel de aplicación.

K3s quedó operativo con un servidor y dos agentes, y la prueba de Nginx confirmó la distribución de réplicas. Las APIs de IA aún no son cargas de K3s, por lo que la arquitectura actual debe describirse como híbrida, esta transparencia delimita el logro alcanzado y establece como siguiente paso natural la migración de los servicios de inferencia al orquestador.

16. Glosario

Término	Definición
API	Interfaz de programación que permite enviar imágenes a los servicios y recibir resultados estructurados.
ARM64	Arquitectura de procesador de 64 bits utilizada por la Raspberry Pi del proyecto.
Clúster	Conjunto de nodos conectados que colaboran para ejecutar servicios y distribuir cargas.
CPU	Unidad central de procesamiento que ejecuta los modelos de inferencia en los tres nodos.
Edge Computing	Procesamiento realizado cerca de la fuente de los datos, dentro de la infraestructura local.
Endpoint	Dirección específica de una API, como /health o /infer, utilizada para solicitar una función.
Fallback	Mecanismo de respaldo que traslada una tarea al nodo maestro cuando el worker no responde.
FPS	Cuadros por segundos procesados y mostrados por el sistema.
HTTP	Protocolo utilizado para transportar las imágenes JPEG y las respuestas entre los nodos.
Inferencia	Ejecución de un modelo entrenado para obtener una predicción a partir de una imagen.
JSON	Formato de texto utilizado por las APIs para devolver clases, confianzas, cajas y métricas.
K3s	Distribución ligera de Kubernetes utilizada para formar y validar el clúster de tres nodos.
Kubernetes	Plataforma de orquestación que administra nodos, contenedores, réplicas y servicios.

Latencia	Tiempo transcurrido desde el inicio del procesamiento hasta la obtención del resultado.
NCNN	Formato y motor de inferencia optimizado para ejecutar modelos en CPU y dispositivos de borde.
Nodo maestro	Raspberry Pi que captura video, coordina solicitudes, presenta el dashboard y ejecuta el respaldo local.
NodePort	Tipo de servicio de Kubernetes que expone una aplicación mediante un puerto accesible en los nodos.
systemd	Administrador de servicios de Linux que inicia y supervisa las APIs de inferencia.
VM	Máquina virtual que funciona como un equipo independiente dentro del clúster.
Worker	Nodo secundario que recibe una imagen y ejecuta una tarea específica de inferencia.
YOLO	Familia de modelos de visión artificial utilizada para la detección de objetos en tiempo real.

17. Referencias

K3s. “Lightweight Kubernetes”. <https://docs.k3s.io/>

K3s. “Quick-Start Guide”. <https://docs.k3s.io/quick-start>

Kubernetes. “Self-healing”. <https://kubernetes.io/docs/concepts/architecture/self-healing/>

Kubernetes. “Nodes”. <https://kubernetes.io/docs/concepts/architecture/nodes/>

Ultralytics. “Image Classification”. <https://docs.ultralytics.com/tasks/classify>

Ultralytics. “NCNN Integration”. <https://docs.ultralytics.com/integrations/ncnn>