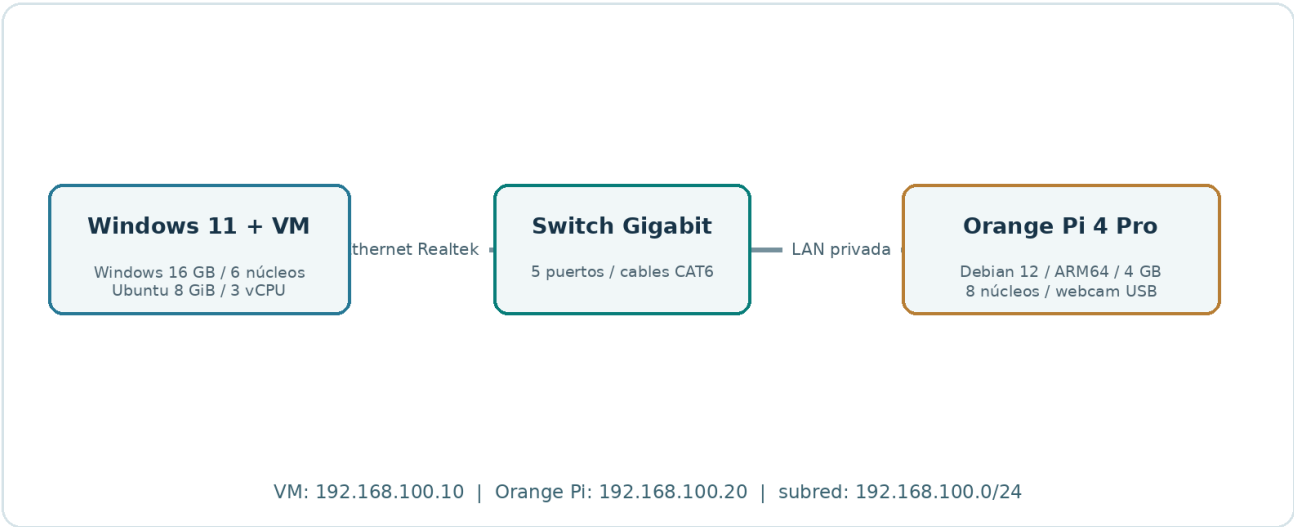


PROYECTO FINAL INTEGRADOR

Diseño, montaje e implementación
de un clúster de Edge Computing
para inferencia de IA local autónoma

EdgeVision · Qwen3-VL + YOLO11-Nano · RPC CPU



Arquitectura física del laboratorio: estación Windows/VM, switch Gigabit y Orange Pi con webcam.

INFORME TÉCNICO · GUÍA OPERATIVA · MATRIZ DE ENTREGABLES

Curso: Arquitectura de Computadoras

Universidad / sección: [Completar con datos del equipo]

Catedrático: [Completar con nombre oficial]

Integrantes: [Agregar nombres y carné de cada integrante]

Fecha de corte de evidencias: 15 de octubre de 2026

Plazo indicado en clase: 16 de octubre de 2026

0. Control del documento y trazabilidad

Finalidad. Registrar de manera verificable la construcción de EdgeVision desde cero, documentar procedimientos reproducibles, presentar la arquitectura y relacionar evidencias con los requisitos oficiales y las indicaciones del catedrático.

CRITERIO DE EVIDENCIA La palabra «verificado» se reserva a hechos sustentados por capturas, registros o resultados confirmados durante el laboratorio. Los pasos redactados como configuración final o procedimiento de cierre que aún no se ejecutaron permanecen marcados «pendiente de aplicar/verificar».

Tipo de dato	Alcance en este informe
Verificado por captura	Instalación Debian ARM64, Ubuntu VM, red Ping/SSH, compilación de llama.cpp en ambas arquitecturas, modelo de texto y asignación de capas RPC0.
Confirmado por responsable	Webcam USB mostró video con retraso, YOLO ejecutó detecciones y permitió capturar un fotograma. No se adjunta una medición de FPS.
Implementado, pendiente de prueba física	Parche 2.0.3 para análisis de fotograma con Qwen3-VL, recuperación local y reintento.
Procedimiento documentado no ejecutado	Persistencia de IP estática, medición iperf3, prueba offline controlada y defensa con resultados completos.
Requisito institucional	PDF de 8 páginas y transcripción de la clase del 10/10/2026, interpretados conjuntamente.

Versión de informe: 1.0. Estado de software documentado: EdgeVision 2.0 con cámara y correcciones 2.0.1/2.0.2 probadas por el usuario; parche 2.0.3 entregado, pendiente de validación en laboratorio.

I. Guía de lectura e índice temático

1. Resumen ejecutivo y necesidad empresarial
2. Objetivos y criterio de éxito
3. Requisitos: PDF frente a clase
4. Fundamentos y selección de tecnología
5. Inventario físico y software
6. Arquitectura de red y componentes
7. Secuencia histórica de implementación
8. Instalación inicial de Orange Pi y VM
9. Red, SSH, Internet y direccionamiento persistente
10. Compilación de llama.cpp y RPC
11. Inferencia local/distribuida y mediciones
12. Implementación del portal Qwen3-VL

13. YOLO-Nano: webcam y detección
 14. Incidencias y parches 2.0.1–2.0.3
 15. Arranque, operación y apagado
 16. Recuperación, continuidad y funcionamiento offline
 17. Estrategia de pruebas y resultados
 18. Seguridad, costos, riesgos y limitaciones
 19. Matriz de cumplimiento de requisitos y rúbrica
 20. Entregables, trabajo colaborativo y defensa
 21. Conclusiones y recomendaciones
- Anexos A–F. Comandos, evidencias, formatos y fuentes

1. Resumen ejecutivo y necesidad empresarial

EdgeVision es un prototipo de **inferencia de inteligencia artificial local** diseñado para demostrar que una empresa puede interpretar imágenes y reconocer objetos sin consumir tokens de servicios comerciales de IA como servicio. El laboratorio utiliza dos arquitecturas de CPU distintas: una Orange Pi 4 Pro (ARM64) y una máquina virtual Ubuntu (x86-64) hospedada en una laptop con Windows 11. Los nodos se conectan por cable mediante un switch Gigabit de cinco puertos.

La solución contiene dos capacidades complementarias dentro del **caso único de visión por computadora**: (a) un detector **YOLO11-Nano** con webcam USB físicamente conectada a la Orange Pi, de acuerdo con el escenario escrito del PDF; y (b) un **modelo multimodal Qwen3-VL-2B-Instruct** que interpreta fotografías subidas desde el navegador o capturadas desde la cámara, de acuerdo con la explicación oral del catedrático. El motor `llama.cpp` dispone de procesamiento remoto por RPC para repartir partes del cálculo entre CPU x86-64 y ARM64.

Se ha validado la compilación en ambos equipos, la conectividad de red, la generación de texto local y con RPC, así como trazas de asignación de capas remotas. El usuario confirmó el video y la captura de la webcam mediante YOLO; la interpretación automática del fotograma capturado aún debe verificarse tras aplicar la versión 2.0.3. El sistema implementa un **respaldo en frío**: si el servidor RPC falla, detiene la instancia distribuida y reconstruye el servicio multimodal solo en la VM; esto no equivale a alta disponibilidad instantánea.

El valor académico central es comprobar capacidad de procesamiento local heterogéneo, documentar sus límites prácticos y presentar una demostración reproducible con monitoreo, resultados observados y procedimiento de continuidad. La propuesta evita costo por token de terceros durante inferencia, aunque **no elimina costos locales** de equipos, electricidad, administración y mantenimiento.

2. Objetivos del proyecto

2.1 Objetivo general

Diseñar, montar, implementar y evaluar un clúster local de Edge Computing de bajo costo y bajo consumo, compuesto por ARM64 y x86-64, capaz de procesar imágenes con IA local cuantizada y detección YOLO-Nano, con evidencia de procesamiento distribuido, acceso web, medición de recursos y recuperación ante la caída del nodo Worker.

2.2 Objetivos específicos

- Montar físicamente Orange Pi, laptop y switch Gigabit y establecer subred privada `192.168.100.0/24`.
- Instalar Debian ARM64 en Orange Pi y Ubuntu Server virtualizado; documentar CPU, RAM, almacenamiento y herramientas.
- Compilar la misma revisión de `llama.cpp` con backend RPC en ambas arquitecturas.
- Ejecutar inferencia local, comparación con RPC y conservar trazas de capas asignadas, tiempos y actividad CPU.
- Implementar un servicio web para subir imágenes y consultar un VLM local mediante Qwen3-VL cuantizado Q4.
- Detectar objetos en tiempo real con webcam USB + YOLO11n ONNX Runtime CPU sobre Orange Pi.
- Probar caída del Worker, modo CPU local de respaldo y operación después de desconectar Internet.
- Preparar informe, evidencias, videos, autoevaluación y evaluación por pares en los formatos de entrega.

2.3 Criterios medibles de aceptación

Criterio	Evidencia esperada	Estado al 15/oct
Conectividad	Ping ambos sentidos, SSH y enlace 1 Gb/s	Ping/SSH y `ethtool` sí; `iperf3` pendiente
CPU distribuidas	Logs RPC0 y actividad de ambas CPU	Asignación RPC0 sí; serie temporal simultánea pendiente
Visión YOLO	Fotograma USB, bounding boxes y FPS	Video/captura confirmados; FPS numérico pendiente
Visión semántica	Respuesta correcta a fotografía capturada	Pendiente validar parche 2.0.3
Falla Worker	Nuevo análisis terminado en CPU VM	Contingencia implementada; prueba de extremo a extremo pendiente
Offline	Portal responde sin acceso a WAN	Pendiente

Criterio	Evidencia esperada	Estado al 15/oct
Entrega	PDF grupal, enlaces videos, Excel evaluaciones	Pendiente incorporar material definitivo

3. Marco de requisitos y diferencias entre fuentes

3.1 Documento escrito oficial (R1)

El PDF define 20 puntos de proyecto: **Fase 1, informe técnico escrito grupal: 10 puntos; Fase 2, defensa técnica en video y coevaluación individual: 10 puntos.** Establece un grupo máximo de cuatro integrantes, un informe formal en PDF para Canvas y un enlace de video individual por estudiante de 5 a 7 minutos. [R1, p. 1].

El alcance técnico escrito exige: nodos embebidos permitidos o mini-PC de bajo consumo; procesadores locales sin servicios de nube pública para inferencia; subnet privada de clase C, IP internas estáticas, interconexión mediante switch Gigabit; tolerancia a caída de un Worker y explicación del middleware de distribución/redistribución. Para visión, enumera explícitamente YOLO ligero o MobileNet y cámara conectada físicamente. [R1, pp. 1–2].

3.2 Instrucciones del catedrático en la clase (R2)

La clase puso el foco en el **problema empresarial**: una organización no quiere depender de pago de tokens y de una solución de IAaaS; por ello debe demostrarse inferencia local por CPU usando equipos cotidianos. El catedrático aceptó una **VM Linux sobre la laptop** conectada por switch con una placa ARM, describió tres casos posibles (chatbot, visión multimodal mediante imagen cargada y what-if) y aclaró que **pfSense no era obligatorio**. [R2, segmento de presentación del proyecto].

También pidió justificar hardware, modelo, motivos de selección, aplicación que resuelve el caso, evidencias de CPU en ambos nodos, profundidad técnica, informe extensivo, videos y participación real de todos los integrantes. Señaló que, si un integrante desconoce el sistema, puede afectar la calificación del equipo. [R2, segmento de rúbrica y defensa].

3.3 Criterio adoptado y cautelas

- **Un solo caso académico:** visión por computadora. YOLO11-Nano aborda literalmente el PDF y Qwen3-VL aporta interpretación semántica a fotografías, según la clase.
- **Procesamiento distribuido:** `llama.cpp` con backend RPC distribuye tensores/operaciones; no “fusiona” automáticamente los núcleos de los sistemas operativos.
- **Hipervisor como segundo nodo:** autorizado verbalmente, aunque el PDF restringe el hardware. Conservar evidencia de la autorización oral y, de ser posible, confirmación escrita.

- **pfSense:** no se implementó. La transcripción lo señala como opcional; las descripciones de la rúbrica PDF contienen términos ajenos al proyecto (WAN dual stack, portal cautivo).
- **Redundancia:** la continuidad frente a caída del Worker es una cosa; la redundancia física de switch/enlace es otra. El diseño cuenta con un solo switch y no es redundante a nivel físico.

4. Fundamentos técnicos y criterios de diseño

4.1 Edge Computing e inferencia autónoma

Edge Computing desplaza procesamiento cerca del origen de datos, en este caso la webcam y el portal LAN. Inferencia significa ejecutar los pesos de un modelo ya entrenado para producir predicciones, no volver a entrenar la red. El prototipo mantiene procesamiento local de CPU; Internet fue necesario para descargar software y pesos, no para consultar modelos de IA externos durante operación.

4.2 LLM, VLM y detección YOLO

Un **LLM** genera lenguaje a partir de secuencias; un **VLM** incorpora un encoder/proyector visual que transforma una imagen en representación consumible por el modelo. Qwen3-VL responde preguntas sobre la fotografía; su salida es descriptiva y probabilística. **YOLO11-Nano** produce categorías, puntuaciones de confianza y cajas delimitadoras por fotograma; no redacta explicaciones amplias sobre la escena. Usar ambos permite clasificar en vivo y describir una captura concreta.

4.3 Cuantización y formato GGUF

La cuantización Q4_K_M reduce el tamaño de almacenamiento y memoria de los pesos respecto a una representación de mayor precisión, a cambio de posibles pérdidas de fidelidad. El archivo GGUF principal se acompaña de un `mmpoj` Q8 para la rama visual. Los tamaños descargados de referencia son aproximadamente 1,11 GB y 445 MB; el consumo de RAM **es mayor** porque incluye runtime, activaciones, caché KV y buffers.

4.4 RPC, partición de capas y recursos

La biblioteca `llama.cpp` utiliza `ggml` para computación tensorial. Su backend RPC expone un dispositivo CPU remoto identificado como `RPC0[192.168.100.20:50052]`. En el modo distribuido, el proceso coordinador en Ubuntu decide qué tensores y operaciones se ejecutan localmente y cuáles en el Worker ARM64. La variable `RPC\_LAYERS=8` establece un objetivo de descarga de capas; **la asignación real se valida con los logs**. La sincronización, transferencia de datos y diferencias de rendimiento pueden hacer que el modo RPC sea más lento con modelos pequeños.

4.5 Disponibilidad y respaldo en frío

No se ejecutan dos modelos multimodales residentes simultáneamente: esto consumiría demasiada RAM. El controlador de Ubuntu dispone de modos `rpc`, `local`, `starting-rpc`, `starting-

local`, `stopped` y `error`; ante pérdida de Worker finaliza la instancia distribuida, carga una local y reintenta según corresponda. Esta estrategia **conserva capacidad de atender solicitudes nuevas** si la VM continúa funcionando, pero requiere tiempo de recarga y no protege contra caída de la VM o el switch.

5. Inventario de hardware y software

5.1 Componentes físicos utilizados

Elemento	Especificación registrada	Uso
Laptop	Windows 11; 16 GB RAM; CPU anunciada de 6 núcleos	Hipervisor VirtualBox, acceso web y nodo x86-64
VirtualBox VM	Ubuntu Server AMD64; 3 vCPU; 8,0 GiB RAM observados; disco planificado 60 GB	Master/coordinador FastAPI + VLM
Orange Pi 4 Pro	4 GB nominal (3,8 GiB visibles); 8 CPU ARM; Debian 12 Bookworm	Worker RPC y detección YOLO
Almacenamiento Orange Pi	Dispositivo MMC de aprox. 116,2 GB observado	Linux ARM64, servicio y ONNX
Switch Gigabit	5 puertos Ethernet	Interconexión local cableada
Cables CAT6	Al menos dos unidades operativas	Unir laptop y Orange Pi al switch
Webcam USB UVC	Modelo no informado; confirmada en funcionamiento	Captura para detector YOLO
Fuente de Orange Pi	Fuente USB-C compatible con requisitos de placa	Alimentación estable
Monitor / teclado / microSD	Usados para instalación inicial y validaciones	Arranque y administración

5.2 Inventario lógico

Software / componente	Tecnología o versión	Responsabilidad
Sistema de placa	Orange Pi Linux 1.0.6 / Debian GNU/Linux 12 Bookworm / aarch64	Base ARM64
Sistema de VM	Ubuntu Server 24.04 LTS AMD64 / x86_64	Base de coordinación
Hipervisor	Oracle VirtualBox 7.x sobre Windows 11	VM y adaptadores virtuales
Motor inferencia	llama.cpp, commit df18bee11e96557b07aa7b12db0976f3c7ac9165	Inferencia CPU y RPC
Modelo de prueba	Qwen2.5-0.5B-Instruct Q4_K_M GGUF (~469 MiB)	Benchmark base

Software / componente	Tecnología o versión	Responsabilidad
texto	visibles)	
VLM	Qwen3-VL-2B-Instruct Q4_K_M + mmproj Q8	Descripción de imágenes
Detección	YOLO11n ONNX, entrada 1×3×480×640	Bounding boxes y clasificación
Runtime detección	ONNX Runtime 1.31.0; OpenCV headless 4.11.0.86; NumPy 1.26.4	Inferencia ARM64 + anotaciones
Web/API	Python, FastAPI, Uvicorn, HTML/CSS/JS, HTTPX, Pillow, psutil	Portal + API de análisis
Gestión procesos	tmux, SSH, scripts Bash y PowerShell	Operación remota
Red y pruebas	NetworkManager, Netplan, ethtool, ping, iperf3, nc, htop	Conectividad, rendimiento y monitoreo

6. Arquitectura física, lógica y direccionamiento

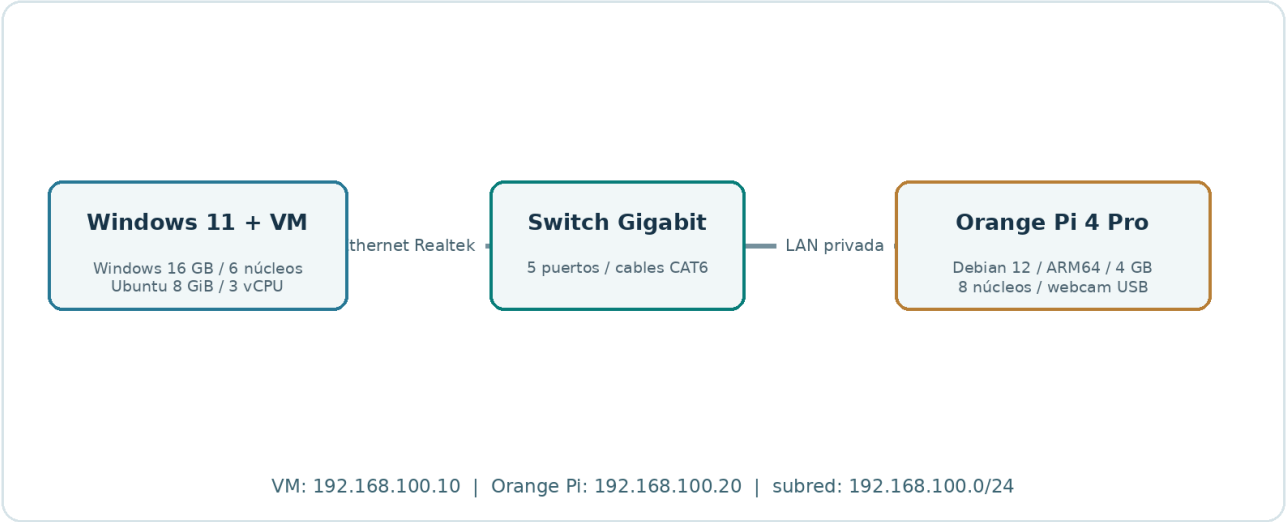


Figura 1. Arquitectura física utilizada para EdgeVision.

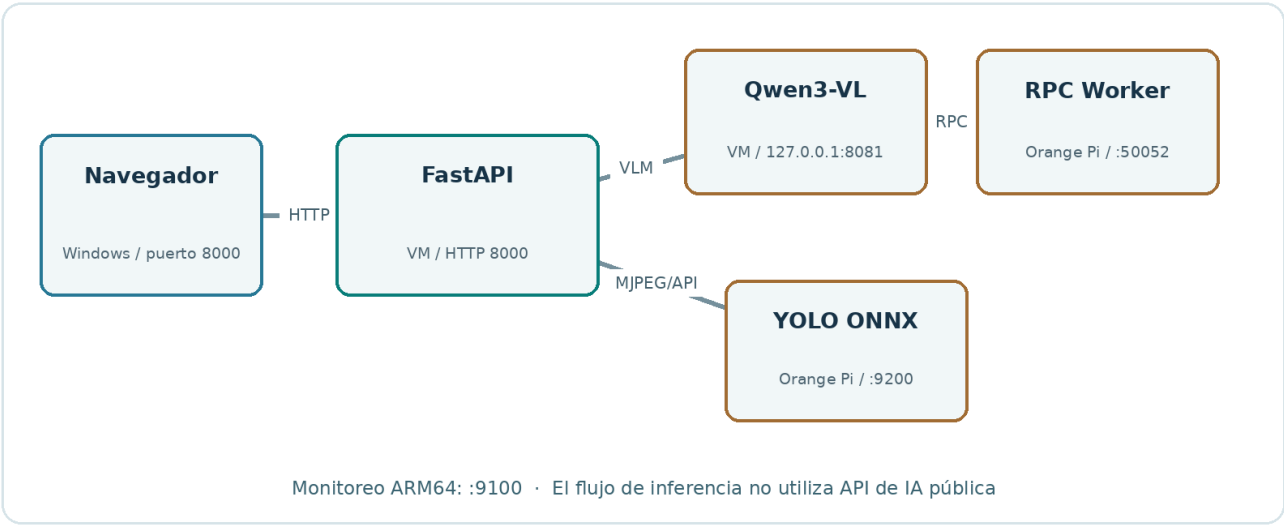


Figura 2. Servicios y puertos del diseño lógico.

6.1 Plan de direccionamiento (subred privada)

Nodo/interfaz	IP/servicio	Función
Ubuntu `enp0s3`	192.168.100.10/24	Master por cable en modo puente Realtek
Orange Pi `end0`	192.168.100.20/24	Worker ARM64 por cable
Windows Ethernet	192.168.137.1/24 al activar ICS	Gateway temporal para descargas
VM `enp0s3` secundaria	192.168.137.128/24 observada	Acceso SSH/portal desde Windows
Orange Pi `end0` secundaria	192.168.137.20/24 observada	Internet compartido de Windows
VM `enp0s8` NAT	10.0.3.15/24 observada	Descargas por segundo adaptador VirtualBox

Las direcciones privadas `192.168.100.10` y `.20` se probaron mediante comandos `ip addr add` y fueron **reales durante el laboratorio**, pero no se registró una prueba de persistencia tras reinicio. El procedimiento final para escribirlas en Netplan/NetworkManager se especifica en la sección 9.

6.2 Tabla de servicios y puertos

Puerto	Origen → destino	Servicio	Observación
22/TCP	Windows/VM → Linux	SSH / SCP	Administración, no publicar en WAN
50052/TCP	VM → Orange Pi	GGML RPC	Backend experimental sin autenticación
8000/TCP	Windows → Ubuntu	Portal FastAPI	Acceso HTTP sin autenticación,

Puerto	Origen → destino	Servicio	Observación
			LAN únicamente
8081/TCP	Ubuntu → Ubuntu	llama-server API OpenAI-compatible	Escucha solo en 127.0.0.1
9200/TCP	VM → Orange Pi	Servicio YOLO webcam	Estado, JPEG, MJPEG, control pausa
9100/TCP	VM → Orange Pi	Métricas ARM64	Mini API de uso CPU/RAM

7. Bitácora cronológica: desde el planteamiento inicial

Etapas	Actividades registradas	Situación resultante
Definición inicial	Se seleccionó interpretación de imágenes por VLM; se revisaron PDF y clase; se priorizó ARM64 + VM heterogénea.	Arquitectura EdgeVision y requisitos identificados.
Preparación placa	Imagen Debian Bookworm para Orange Pi; grabación en tarjeta; arranque; cambio contraseña; instalación SSH y utilidades.	Debian ARM64 y 8 núcleos verificados.
VM Ubuntu	VirtualBox, Ubuntu Server x86_64, 3 vCPU, RAM observada 8,0 GiB y dos adaptadores.	Nodo Ubuntu preparado.
Problema red puente	Adaptador 1 apuntaba a MediaTek Wi-Fi; se corrigió hacia Realtek Gaming 2.5GbE.	Conectividad por switch y SSH.
IP y acceso WAN	Direcciones temporales .100.10/.100.20; Windows compartió Wi-Fi a Ethernet; .137.1 gateway; se corrigió reloj NTP.	Ping privado 4/4 y APT operativo.
Compilación Ubuntu	CMake inicial sin RPC; error “unknown target ggml-rpc-server”; se activó GGML_RPC=ON y compilaron binarios.	llama-server, llama-cli, ggml-rpc-server presentes.
Compilación Orange Pi	Se usó commit idéntico df18bee11..., `--parallel 2` para ARM64.	Worker RPC generado y puerto 50052 accesible.
Modelos de texto	Qwen2.5 0.5B Q4; CLI local y RPC, logs RPC0.	10,4 vs 8,6 tok/s, evidencia de offload.
Portal multimodal	Se integró FastAPI, Qwen3-VL, mmpoj, monitor, watchdog y respaldo CPU local.	Portal mostró cámara y estados; VLM tuvo errores en análisis de captura.
Visión con webcam	Se agregó YOLO11n ONNX y pestaña Cámara; cámara USB conectada a placa.	Captura y video observados, con lag.
Parche 2.0.1	SHA256 esperado no coincidía con revisión ONNX; se corrigió lista permitida.	Modelo verificado; se avanzó al runtime.

Etapas	Actividades registradas	Situación resultante
Parche 2.0.2	El código suponía entrada cuadrada; ONNX exigía [1,3,480,640].	Detector soportó tamaño rectangular; webcam funcionó según usuario.
Parche 2.0.3	La acción capturar y analizar podía fallar por VLM no listo; se entregó recuperación local y reintento.	Código y pruebas del parche preparados; validación física pendiente.

Esta bitácora distingue **resoluciones confirmadas** (red, compilación, YOLO/cámara) de correcciones de software **entregadas pero aún no comprobadas físicamente** (parche 2.0.3). No constituye una declaración de que se hayan grabado ya los videos o configurado direcciones IP persistentes.

8. Instalación inicial: Orange Pi y Ubuntu virtual

8.1 Preparación y arranque Debian ARM64

1. Identificar la placa como Orange Pi 4 Pro, versión de 4 GB; obtener imagen Debian Bookworm ARM64 específica de ese modelo desde el sitio de soporte del fabricante.
2. Grabar el archivo `.img` en microSD mediante balenaEtcher desde Windows 11; verificar la unidad seleccionada para no borrar el disco de la laptop.
3. Montar tarjeta, Ethernet CAT6, HDMI/teclado y fuente USB-C adecuada; iniciar y cambiar contraseña del usuario `orangepi`.
4. Activar SSH para administración posterior y registrar el estado de CPU, RAM, interfaces y almacenamiento.

```
uname -m
cat /etc/os-release
free -h
nproc
lsblk -o NAME,SIZE,TYPE,MOUNTPOINTS
ip -br addr
passwd
sudo apt update
sudo apt install -y openssh-server htop curl git iperf3 ethtool tmux sysstat
sudo systemctl enable --now ssh
systemctl is-active ssh
```

Resultados documentados: `uname -m = aarch64`; Orange Pi 1.0.6 Bookworm basado en Debian 12; RAM visible ~3,8 GiB; 8 procesadores lógicos; almacenamiento MMC ~116,2 GB; `end0` como Ethernet. Se observó swap zram ~1,9 GiB; es memoria comprimida/virtual, no RAM física adicional.

8.2 Configuración de VM VirtualBox

5. Descargar VirtualBox para Windows y la ISO Ubuntu Server 24.04 LTS AMD64.

6. Crear VM EdgeVision-Master con 3 vCPU; memoria observada 8 GiB; disco VDI dinámico objetivo 60 GB.
7. Adaptador 1: **Bridged Adapter**, dispositivo anfitrión **Realtek Gaming 2.5GbE Family Controller**, tarjeta virtual Intel PRO/1000 MT Desktop.
8. Adaptador 2: **NAT**, disponible para paquetes y descarga de pesos; se puede desconectar para la prueba sin Internet.
9. Instalar Ubuntu Server sin escritorio, usuario de laboratorio `milo`, paquete OpenSSH Server y actualizaciones esenciales.

```
uname -m
nproc
free -h
ip -4 -br addr
ip route
sudo apt update && sudo apt upgrade -y
sudo apt install -y openssh-server netcat-openbsd tmux
```

Resultados observados: `x86\_64`, 3 vCPU, `free -h` mostró ~8,0 GiB de RAM; interfaces `enp0s3` (puente) y `enp0s8` (NAT). El valor de 7 GiB fue una **recomendación preliminar**, no la memoria finalmente mostrada en las capturas.

9. Red, Internet compartido, SSH e IP estática

9.1 Diagnóstico y recuperación de la red cableada

Al principio las pruebas ICMP fallaron porque el puente del Adaptador 1 seleccionaba **MediaTek Wi-Fi 6 MT7921**. Se corrigió a **Realtek Gaming 2.5GbE**, que está físicamente unida al switch. La Orange Pi mostró `Speed: 1000Mb/s`, `Duplex: Full` y `Link detected: yes`; luego Ubuntu obtuvo comunicación con el Worker. Al reiniciar la red, Ubuntu perdió temporalmente la IP `.100.10`, que se recuperó con `ip addr add`.

```
# Orange Pi
sudo ethtool end0 | grep -E 'Speed:|Duplex:|Link detected:'
ip -4 -br addr show end0
sudo ip addr add 192.168.100.20/24 dev end0

# Ubuntu VM
sudo ip addr add 192.168.100.10/24 dev enp0s3
ip route get 192.168.100.20
ping -I 192.168.100.10 -c 4 192.168.100.20
ssh orangepi@192.168.100.20
```

Se obtuvo una prueba de cuatro pings, cuatro recibidos, **0 % pérdida** y latencia aproximada entre 1,5 y 2,3 ms en la captura de Ubuntu. Esto demuestra conectividad IP, no por sí mismo rendimiento Gigabit sostenido; `iperf3` sigue siendo la prueba cuantitativa de caudal.

9.2 Internet compartido exclusivamente para aprovisionamiento

En Windows 11 se activó **Internet Connection Sharing (ICS)** de la conexión Wi-Fi hacia Ethernet. Windows usó `192.168.137.1`; la Orange Pi recibió manualmente `192.168.137.20` en la misma interfaz `end0` y la ruta predeterminada hacia el gateway. Se observó ping a 1.1.1.1 (3/4

respuestas) y resolución DNS de GitHub; tras corregir la fecha de la placa, `apt update` terminó sin bloqueo.

```
# Orange Pi: IP secundaria y gateway temporal
sudo ip addr add 192.168.137.20/24 dev end0
sudo ip route replace default via 192.168.137.1 dev end0
ping -c 4 192.168.137.1
ping -c 4 1.1.1.1
getent hosts github.com
sudo timedatectl set-timezone America/Guatemala
sudo timedatectl set-ntp true
timedatectl status
sudo apt update
```

9.3 Propuesta de IP persistentes: procedimiento de cierre por aplicar

ESTADO En el laboratorio se utilizaron direcciones IP reales, pero asignadas transitoriamente con `ip addr add`. La persistencia después de reinicio no está acreditada; se documenta a continuación la configuración objetivo que debe aplicarse y comprobarse desde consola local para no cortar SSH.

Ubuntu (Netplan): primero identificar el YAML efectivo con `sudo netplan get` y `ls /etc/netplan/`. Respalidar el archivo antes de editar. Una propuesta compatible con dos interfaces es la siguiente; debe integrarse con el archivo real sin duplicar definiciones:

```
# Ejemplo de contenido NETPLAN (NO pegar sin revisar YAML existente):
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: true
      addresses: [192.168.100.10/24]
      dhcp4-overrides: {route-metric: 200}
      optional: true
    enp0s8:
      dhcp4: true
      dhcp4-overrides: {route-metric: 50}
      optional: true
```

```
# Validación desde consola de VirtualBox:
sudo netplan generate
sudo netplan try
ip -4 -br addr
ip route get 192.168.100.20
```

Orange Pi (NetworkManager): identificar primero nombre de perfil activo. Es seguro mantener Internet ICS por DHCP y declarar una dirección adicional estática `192.168.100.20/24` en el mismo perfil si el administrador admite esa combinación. Ejemplo orientativo, sustituyendo `` por el valor real:

```
nmcli -f NAME,UUID,TYPE,DEVICE connection show
nmcli -f NAME,DEVICE connection show --active
# Sustituir el nombre; ejecutar desde consola física:
sudo nmcli con mod '<PERFIL_ETHERNET>' ipv4.method auto +ipv4.addresses 192.168.100.20/24
sudo nmcli con up '<PERFIL_ETHERNET>'
ip -4 -br addr show end0
ping -c 4 192.168.100.10
```

Si `end0` aparece conectado externamente y no existe un perfil NetworkManager asociado, debe crearse/importarse uno bajo la administración real de la imagen Orange Pi; **no ejecutar a ciegas**

`nmcli con up` desde la única sesión remota. La aceptación exige reiniciar ambos sistemas, comprobar `.100.10` y `.100.20``, probar ping y SSH en ambos sentidos y documentar capturas.

9.4 Acceso SSH y automatización desde Windows 11

El usuario evitó introducir secuencias largas directamente en la VM usando SSH de Windows y scripts de PowerShell/BAT. El acceso se realizó con la IP de Ubuntu en el segmento ICS; alternativamente se configuró redirección NAT de puerto local ``127.0.0.1:2222`` hacia SSH 22 de Ubuntu.

```
# PowerShell de Windows:
ssh milo@192.168.137.128
# Alternativa con NAT Port Forwarding de VirtualBox:
ssh -p 2222 milo@127.0.0.1
# Tras acceder a Ubuntu:
ssh orangepi@192.168.100.20
```

9.5 Prueba de velocidad pendiente de registrar

```
# Orange Pi (servidor):
iperf3 -s
# Ubuntu (cliente):
iperf3 -c 192.168.100.20 -t 15
iperf3 -c 192.168.100.20 -R -t 15
```

Registrar el throughput medido en Mbit/s de ida y vuelta; no consignar 900–940 Mbit/s como resultado propio hasta que ``iperf3`` lo mida. El enlace negociado a 1000Mb/s no equivale necesariamente al caudal útil.

10. Compilación exacta de llama.cpp para x86-64 y ARM64

10.1 Instalación de dependencias y código

```
sudo apt update
sudo apt install -y git cmake ninja-build build-essential libcurl4-openssl-dev libssl-dev pkg-config
ccache
cd ~
git clone https://github.com/ggml-org/llama.cpp.git
cd ~/llama.cpp
git rev-parse HEAD
```

En Ubuntu, la revisión confirmada fue ``df18bee11e96557b07aa7b12db0976f3c7ac9165``. En Orange Pi se hizo ``git checkout`` a ese mismo hash, para evitar diferencias en protocolo y operaciones RPC. La compilación en la placa se realizó limitando concurrencia para sus 4 GB de memoria.

10.2 Habilitar RPC y compilar en Ubuntu

El primer intento compiló ``llama-server`` y ``llama-cli``, pero el objetivo ``ggml-rpc-server`` no existía porque la caché CMake mostraba ``GGML_RPC:BOOL=OFF``. Se corrigió **sin borrar el repositorio** activando ``GGML_RPC=ON`` y ``LLAMA_BUILD_TOOLS=ON``.

```
cd ~/llama.cpp
cmake -S . -B build -G Ninja -DCMAKE_BUILD_TYPE=Release -DGGML_RPC=ON -DGGML_NATIVE=ON
-DLLAMA_BUILD_TOOLS=ON
cmake --build build --parallel 3 --target llama-server llama-cli ggml-rpc-server
```

```
ls -lh build/bin/llama-server build/bin/llama-cli build/bin/ggml-rpc-server
grep -E '^((GGML_RPC|LLAMA_BUILD_TOOLS):' build/CMakeCache.txt
```

10.3 Compilar worker en Orange Pi

```
ssh orangepi@192.168.100.20
cd ~
git clone https://github.com/ggml-org/llama.cpp.git
cd ~/llama.cpp
git checkout df18bee11e96557b07aa7b12db0976f3c7ac9165
cmake -S . -B build -G Ninja -DCMAKE_BUILD_TYPE=Release -DGGML_RPC=ON -DGGML_NATIVE=ON
-DLLAMA_BUILD_TOOLS=ON
cmake --build build --parallel 2 --target ggml-rpc-server
ls -lh build/bin/ggml-rpc-server
```

El resultado observado fue el enlace final `build/bin/ggml-rpc-server` de aproximadamente 179 KB. Las pruebas individuales de extensiones ARM que mostraban “Failed” eran detección de capacidades opcionales (`SVE`, `SME`), no fallos de CMake, pues el objetivo compiló con éxito.

10.4 Arranque del servicio de procesamiento remoto

```
# En Orange Pi, iniciar en tmux:
tmux new -s edge-rpc
cd ~/llama.cpp
./build/bin/ggml-rpc-server --host 192.168.100.20 --port 50052 --threads 4
# Soltar tmux: Ctrl+B, luego D

# Comprobar desde Ubuntu:
nc -vz -w 3 192.168.100.20 50052
```

La comprobación devolvió `Connection to 192.168.100.20 50052 ... succeeded!`. El puerto abierto prueba transporte TCP; la carga real de tensores se confirmó posteriormente con registros del modelo.

11. Inferencia local y pruebas RPC: resultados medidos

11.1 Modelo de calibración textual

Antes de cargar el VLM multimodal se utilizó **Qwen2.5-0.5B-Instruct Q4_K_M GGUF** de ~469 MiB visible en disco para reducir tiempo, memoria y complejidad de los primeros ensayos. Esto permitió separar errores de conectividad y RPC de los posibles errores del proyector visual.

```
mkdir -p ~/models
curl -fL --retry 3 -o ~/models/qwen2.5-0.5b-instruct-q4_k_m.gguf 'https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct-GGUF/resolve/main/qwen2.5-0.5b-instruct-q4_k_m.gguf'
```

11.2 Ejecución por CPU local Ubuntu

```
cd ~/llama.cpp
./build/bin/llama-cli -m ~/models/qwen2.5-0.5b-instruct-q4_k_m.gguf -p 'Explica que es Edge Computing en tres frases.' -n 80 -c 1024 -t 3 -ngl 0
```

11.3 Ejecución con dispositivo remoto RPC

```
./build/bin/llama-cli -m ~/models/qwen2.5-0.5b-instruct-q4_k_m.gguf --rpc 192.168.100.20:50052 -ngl 8 -t 3 -c 1024 -n 80 -p 'Explica que es Edge Computing en tres frases.'
```

Prueba	Resultado observado	Interpretación
Solo VM Ubuntu	Generación ~10,4 tokens/s	Rendimiento CPU local en este ensayo
VM con RPC Orange Pi	Generación ~8,6 tokens/s	Ejecución con backend remoto habilitado
Variación relativa	$(8,6 / 10,4 - 1) \times 100 \approx -17,3 \%$	La distribución fue más lenta con el modelo 0.5B
Registro de dispositivos	`RPC0[192.168.100.20:50052]`	Backend remoto reconocido
Caché y buffers	Capas 17–23 en RPC0; compute buffer RPC 33,01 MiB; CPU 38,27 MiB	Asignación de memoria y cómputo repartida

Los porcentajes de CPU del tablero durante la recuperación del VLM (por ejemplo 100 % Ubuntu y 25 % Orange Pi) **no prueban distribución en esa solicitud**, porque la ruta observada fue “Solo VM”. Para afirmar cálculos remotos hace falta combinar logs RPC0 y actividad CPU al mismo tiempo.

11.4 Trazas conservadas

```
# Resumen de logs a conservar en el informe:
llama_kv_cache: layer 17: dev = RPC0
...
llama_kv_cache: layer 23: dev = RPC0
RPC0[192.168.100.20:50052] compute buffer size = 33.01 MiB
CPU compute buffer size = 38.27 MiB
# Al usar --rpc, comprobar asignación en el registro completo.
```

12. Diseño e implementación del portal multimodal Qwen3-VL

12.1 Componentes y archivos

El portal se implementó como una aplicación FastAPI que presenta HTML/CSS/JavaScript al navegador Windows. `backend/main.py` recibe solicitudes, valida imágenes, recoge métricas y envía un payload compatible con la API de chat de `llama-server`. `backend/engine.py` controla el único proceso residente de Qwen3-VL, sus modos RPC/local, vigilancia y logs. `scripts/start.sh` ejecuta el portal en una sesión `tmux` y `scripts/stop.sh` lo apaga de forma ordenada.

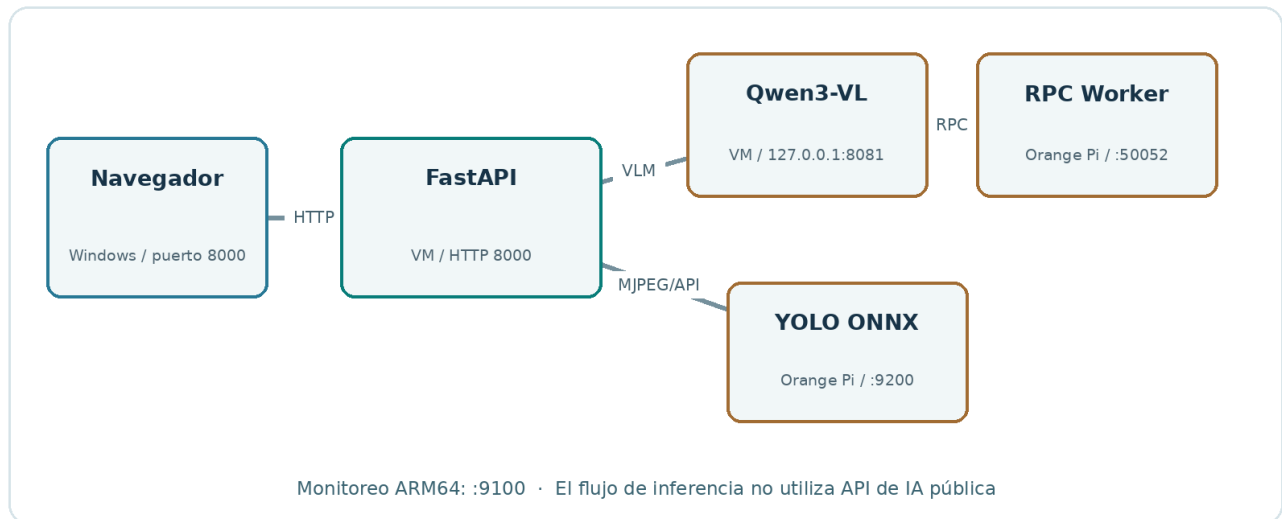


Figura 3. Separación entre navegador, API del portal, modelo local y servicios ARM64.

12.2 Descarga de pesos multimodales

```

cd ~/edge-vision-lab
bash scripts/install_vm.sh
# Equivale a descargar en ~/edge-vision-models:
# Qwen3VL-2B-Instruct-Q4_K_M.gguf
# mmproj-Qwen3VL-2B-Instruct-Q8_0.gguf
ls -lh ~/edge-vision-models/*.gguf

```

En el portal, la configuración actual conocida del repositorio utiliza `VM\_THREADS=3`, `RPC\_LAYERS=8`, `CONTEXT\_TOKENS=4096`, `IMAGE\_MAX\_TOKENS=512`, límite de una solicitud simultánea en la instancia del modelo (`-np 1`) y buffers de lote `512`. Esta es la configuración del código final; no debe confundirse con una sugerencia preliminar de contexto 1024 usada en las primeras pruebas CLI.

12.3 Flujo de validación y análisis

10. Usuario selecciona una imagen JPG/PNG/WebP de hasta 5 MB y formula una instrucción de 1 a 1000 caracteres.
11. FastAPI verifica que el archivo sea realmente imagen, limita 6 megapíxeles, normaliza orientación EXIF, redimensiona a máximo 640×640 y codifica JPEG optimizado.
12. Se construye una solicitud JSON de chat que contiene texto y `image\_url` en formato `data:image/jpeg;base64,...`.
13. El portal solicita a `http://127.0.0.1:8081/v1/chat/completions`, con `max\_tokens=180`, `temperature=0.1` y modo no streaming.
14. El motor genera una descripción y el portal presenta respuesta, modo (`rpc-primary`/`vm-fallback`) y duración de consulta.
15. Se registra fecha UTC, prefijo del hash SHA-256 de la imagen, modo, latencia y cambios de respaldo; no se persiste la fotografía original en ese log.

```

# Consulta de salud y estado del portal:
curl --noproxy '*' http://127.0.0.1:8000/api/status
# URL del navegador en red local:

```

```
http://192.168.137.128:8000
# Endpoint de análisis (form-data multipart):
POST /api/analyze
  image: archivo JPG/PNG/WebP
  question: texto de consulta
```

12.4 Endpoints de aplicación documentados

Ruta HTTP	Operación	Objeto
GET /	Abre el portal	Página index.html
GET /api/status	Estado VM, RPC, VLM, cámara y métricas	Panel
POST /api/analyze	Analiza archivo + instrucción	Qwen3-VL
POST /api/use-rpc	Reactivar procesamiento distribuido	Motor
POST /api/use-local	Cambiar a solo VM	Motor
GET /api/camera/status	Estado de cámara remota	YOLO
GET /api/camera/snapshot	Fotograma original JPEG	Captura
GET /api/camera/stream	Relé MJPEG mismo origen	Video web
POST /api/camera/pause resume	Pausar/reanudar captura/inferencia	Arbitraje de CPU

13. YOLO11-Nano: webcam física y detección en el borde

13.1 Arquitectura de visión en tiempo real



Figura 4. Captura USB en Orange Pi, inferencia YOLO en CPU ARM64 y visualización web.

La webcam se conecta físicamente a la Orange Pi y la aplicación Linux utiliza OpenCV con interfaz V4L2 para obtener fotogramas. Se solicita una resolución de captura de **640×480**, formato MJPEG cuando está disponible y búfer pequeño para reducir retraso. El detector ONNX recibe tensores **[1, 3, 480, 640]** conforme a la exportación usada, procesa clases de objetos, aplica filtrado por confianza y supresión no máxima (NMS), dibuja recuadros/etiquetas y construye la vista MJPEG.

13.2 Preprocesamiento, posprocesamiento y métricas

- **Preprocesamiento:** adecuación del fotograma al tamaño `(alto, ancho)` de entrada del modelo, normalización y organización de canales NCHW.
- **Predicción:** ONNX Runtime CPUExecutionProvider calcula el mapa de detecciones de YOLO11n con dos hilos de inferencia y optimizaciones habilitadas.
- **NMS:** elimina propuestas superpuestas por categoría y conserva las detecciones relevantes, mostrando clase COCO traducida y confianza.
- **Anotación:** OpenCV dibuja cajas, etiquetas y FPS sobre una copia del fotograma.
- **Instantánea original:** el proceso conserva una versión JPEG **sin recuadros**, que se usa al elegir Capturar/Analizar con Qwen3-VL.
- **Panel:** expone FPS real, tiempo de inferencia en milisegundos, total de fotogramas, pausa, antigüedad del último frame y último error.

13.3 Restricciones para Orange Pi de 4 GB

Se fijó `CAMERA\_TARGET\_FPS=1`, `CAMERA\_ONNX\_THREADS=2`, umbral de confianza inicial 0,40 y una sola instancia del detector. El usuario observó que la cámara **funciona con cierto lag**; no hay una medición FPS numérica final. Una webcam UVC de otra resolución, baja iluminación, CPU compartida con RPC o tráfico/serialización MJPEG pueden aumentar la latencia.

13.4 Puertos y operaciones de cámara

```
# En Orange Pi:
ls /dev/video*
v4l2-ctl --list-devices
cd ~/edgevision-camera
bash start.sh
# Servicio de cámara en puerto 9200 (solo interfaz privada)

# En Ubuntu:
curl --noproxy '*' http://192.168.100.20:9200/status
# Se muestran video y captura vía:
# GET /api/camera/stream --> GET Pi:9200/stream.mjpg
# GET /api/camera/snapshot --> GET Pi:9200/snapshot.jpg
```

Para evitar competencia de procesamiento ARM64, la integración pausa temporalmente YOLO cuando se envía una captura a Qwen3-VL. Esta pausa reduce carga pero introduce discontinuidad intencionada en el video, que debe explicarse en la defensa.

14. Incidencias, análisis de causa raíz y parches

Incidencia	Síntoma observado	Corrección / estado
Puente Wi-Fi erróneo	Ping/SSH fallaban con Destination Host Unreachable	Seleccionar Realtek Ethernet en Adaptador 1 VirtualBox; IP privadas correctas.
No existía objetivo RPC	Ninja: unknown target ggml-rpc-server	Reconfigurar GGML_RPC=ON y LLAMA_BUILD_TOOLS=ON.
Internet y tiempo placa	Network unreachable, Release file not valid yet	ICS de Wi-Fi Windows y NTP/corrección de reloj.
Escritura script bash	`invalid argument: bash` en `llama-cli`	Sustituir archivo con heredoc correcto y validar con `bash -n`.
YOLO inicial	`yolo11n.onnx.part: FAILED` SHA-256	Parche 2.0.1 verificó hash de revisión descargada.
YOLO ONNX rectangular	Modelo exige [1,3,480,640], código asumía [1,3,S,S]	Parche 2.0.2 admite entrada rectangular, dimensiona/reescala cajas.
Análisis de captura	El portal capturó imagen, pero falló envío/uso de Qwen3-VL	Parche 2.0.3 implementado; pendiente probar en VM real.

14.1 Parche 2.0.1: integridad del archivo ONNX

El instalador descargó el modelo, pero el checksum esperado correspondía a otro contenido; el proceso se abortó de forma segura. El nuevo checksum reconocido fue ``baeb4c1d6f3bb460689823cc2d40daa31e6156f2517fcd48aab0db9490d9f4e5`` para el ONNX utilizado. La comprobación SHA-256 se mantuvo activa y el archivo se copió a la placa una vez validado.

14.2 Parche 2.0.2: entrada rectangular

El error completo ``ValueError: El modelo debe aceptar entrada estatica [1,3,S,S], recibido [1,3,480,640]`` provenía de validar indebidamente que ancho y alto fueran iguales. Se corrigieron la lectura de dimensiones, el preprocesamiento y el remapeo de cajas a la resolución original. Después de este parche, el usuario informó que el video de webcam **se veía y permitía capturar**; quedaba cierta latencia.

14.3 Parche 2.0.3: captura hacia VLM

Al seleccionar “Capturar y analizar con Qwen3-VL”, el usuario recibió un mensaje de fallo; previamente el dashboard había mostrado Qwen3-VL “No disponible”. El código del parche 2.0.3 añade un intento controlado de restaurar el modelo en CPU Ubuntu cuando ``ensure_ready()`` falla; no toca cámara, YOLO, RPC ni los GGUF. Conserva la fotografía y ofrece reintento local en la

interfaz, además de logs y copia de seguridad. **A la fecha del informe, el usuario había recibido el parche pero aún no lo había probado físicamente.**

14.4 Gestión segura de cambios

Las reparaciones no sobrescribieron deliberadamente la instalación completa: transferían pocos archivos, comprobaban SHA-256, detenían solo los servicios afectados y mantenían respaldos. El parche 2.0.3, por ejemplo, valida hashes conocidos de `backend/main.py` e `index.html`, crea `evidencias/respaldo\_integracion\_203\_<fecha>/` y revierte los dos archivos si el portal no arranca. Esto evita degradar los elementos ya funcionales.

15. Manual operativo: cómo levantar EdgeVision desde apagado

PRERREQUISITO Los comandos de esta guía asumen que la instalación de paquetes y modelos ya está hecha. Las IP del clúster deben existir; hasta completar la persistencia, verificar/agregar temporalmente la dirección correspondiente después de reiniciar.

15.1 Secuencia de encendido y comprobación

16. **Paso 1 – física:** conectar placa, webcam USB, cable CAT6 al switch, laptop por puerto Realtek, fuente estable y encender el switch.
17. **Paso 2 – sistemas:** iniciar Orange Pi Debian; abrir VirtualBox en Windows e iniciar VM Ubuntu Server.
18. **Paso 3 – red:** verificar `192.168.100.20`/`192.168.100.10`, ping y `ssh` desde Ubuntu a placa. El acceso a Internet no es requisito de inferencia una vez descargados los modelos.
19. **Paso 4 – RPC:** comprobar `tmux ls` en placa; iniciar sesión `edge-rpc` si no existe y comprobar `nc -vz` desde Ubuntu.
20. **Paso 5 – YOLO:** verificar cámara `/dev/video\*`, sesión `edgevision-camera` y endpoint `http://192.168.100.20:9200/status`.
21. **Paso 6 – portal:** iniciar `bash ~/edge-vision-lab/scripts/start.sh` desde Ubuntu o `Iniciar-EdgeVision.bat` desde Windows.
22. **Paso 7 – salud:** consultar `/api/status`; esperar que el VLM se cargue y probar una imagen antes de la defensa.

```
# En Orange Pi:
ip -4 -br addr show end0
tmux ls
# Si no hay sesión edge-rpc:
tmux new-session -d -s edge-rpc    '$HOME/llama.cpp/build/bin/ggml-rpc-server --host 192.168.100.20 --port 50052 --threads 4'
# Si no hay sesión edgevision-camera:
tmux new-session -d -s edgevision-camera    'bash $HOME/edgevision-camera/start.sh >> $HOME/edgevision-camera/camara.log 2>&1'

# En Ubuntu:
nc -vz -w 3 192.168.100.20 50052
bash ~/edge-vision-lab/scripts/start.sh
bash ~/edge-vision-lab/scripts/status.sh
curl --noproxy '*' http://127.0.0.1:8000/api/status
```

Los comandos de arranque ``tmux new-session`` deben utilizarse **solo si no existe la sesión**; iniciar dos instancias en los mismos puertos causa conflictos. Si el comando de sesión genera error o expande incorrectamente ``$HOME``, usar la versión explícita de script ``cd ~/llama.cpp && ./build/bin/ggml-rpc-server ...`` dentro de una sesión interactiva.

15.2 Operación normal del portal

23. Abrir URL local ``http://192.168.137.128:8000`` mientras siga vigente esa IP; si cambió, leer IP real en Ubuntu.
24. Seleccionar **Imagen · Qwen3-VL** para subir JPG/PNG/WebP y escribir una pregunta, o **Cámara en vivo · YOLO-Nano** para mostrar recuadros.
25. Pulsar Capturar fotograma para obtener JPEG desde la Orange Pi; al analizar, la imagen original es procesada por Qwen3-VL.
26. Revisar panel: estado Worker, VLM, webcam; ruta “RPC + VM” o “Solo VM”; CPU/RAM y recuento de recuperaciones.
27. Evitar múltiples consultas simultáneas: ``request_lock`` serializa análisis, el servidor usa ``-np 1`` y CPU es limitada.

15.3 Apagado seguro y mantenimiento

```
# En Ubuntu: detener portal y VLM juntos
bash ~/edge-vision-lab/scripts/stop.sh
# En Orange Pi: detener solo si se requiere
# (cámara y RPC pueden permanecer durante uso del sistema):
tmux kill-session -t edgevision-camera
tmux kill-session -t edge-rpc
# Apagar cada Linux correctamente desde su consola:
sudo poweroff
```

Para revisar fallas sin romper el entorno: ``bash ~/edge-vision-lab/scripts/status.sh``, ``tail -n 80 ~/edge-vision-lab/evidencias/modelo-vm.log``, ``tail -n 80 ~/edgevision-camera/camara.log``, ``tmux ls``, ``ss -ltnp`` y ``journalctl -xe``.

16. Respaldo del Worker, continuidad y prueba sin Internet

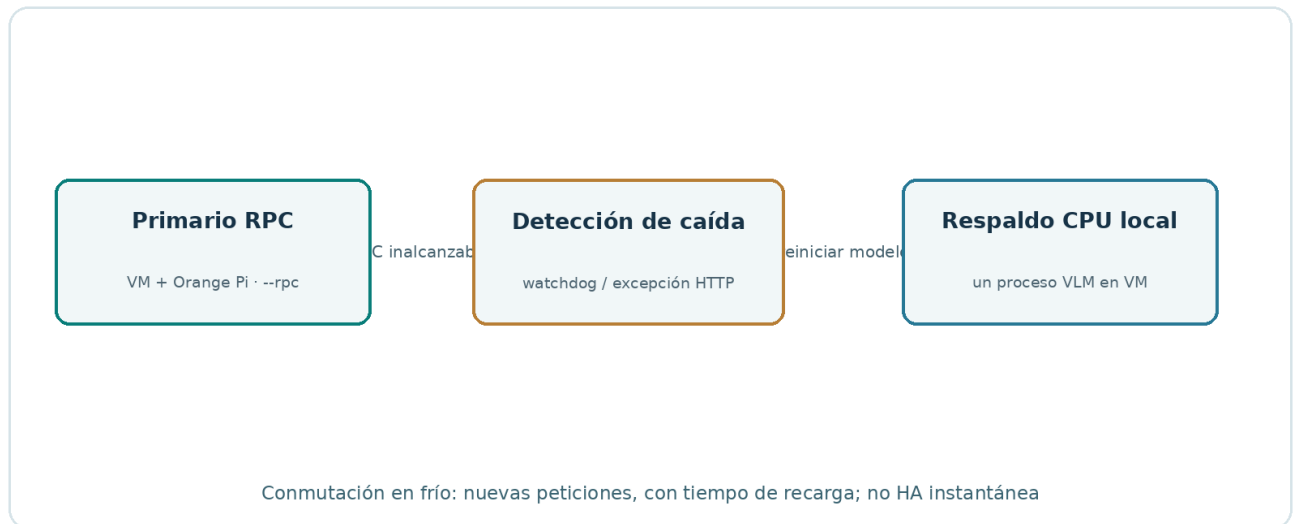


Figura 5. Lógica de conmutación en frío desde procesamiento RPC a CPU local.

16.1 Política de conmutación de inferencia

El watchdog revisa el alcance TCP del RPC aproximadamente cada cinco segundos. Si falla mientras está en modo RPC, el controlador apaga la instancia del `llama-server` distribuido y arranca la misma pareja GGUF/mmproj con `-ngl 0` en Ubuntu; para ahorrar RAM utiliza un proceso único. Cuando el servidor local responde a `/health`, se atienden nuevas solicitudes. Una petición interrumpida puede necesitar reejecución desde el inicio y el retorno a RPC se inicia manualmente desde el portal.

16.2 Protocolo de prueba controlada (pendiente de ejecutar completo)

28. Asegurar previamente una inferencia exitosa en modo `rpc-primary` y guardar captura/log.
29. En Orange Pi simular fallo únicamente del Worker ejecutando `tmux kill-session -t edge-rpc`.
30. Observar la transición del panel a `Solo VM`; esperar recarga de Qwen3-VL (puede tardar minutos).
31. Enviar una **nueva** imagen y comprobar una respuesta completa en `vm-fallback`; registrar tiempo y número de recuperaciones.
32. Reactivar Worker y pulsar «Reactivar RPC»; validar otra respuesta y estado restaurado.
33. No declarar “alta disponibilidad sin interrupción”: se documenta **tolerancia funcional con respaldo frío** y limitaciones.

16.3 Prueba aislada de Internet (pendiente de demostración completa)

Los pesos y paquetes se descargan con Internet durante instalación; en ejecución se requiere solo la LAN. Para probarlo debe cortarse **toda salida WAN** (Wi-Fi Windows y otras rutas/NAT que comuniquen a Internet), manteniendo Ethernet privada. Verificar con `ping -c 2 1.1.1.1` que la WAN no responde y con `ping -c 2 192.168.100.20` que el Worker sí responde. Luego analizar una

fotografía o detectar objetos y conservar evidencias. No basta con apagar solo NAT si Windows sigue compartiendo Wi-Fi.

17. Plan de pruebas, datos y resultados

17.1 Resultados demostrados vs pruebas pendientes

ID	Prueba	Criterio de aprobación	Situación
T01	Sistema ARM64	Debian/aarch64, 8 CPU, RAM visible	Verificada
T02	Ubuntu VM	x86_64, 3 vCPU y RAM visible	Verificada
T03	Enlace físico	`ethtool` 1000Mb/s + Ping/SSH	Verificada; throughput pendiente
T04	Puertos RPC	TCP 50052 accesible	Verificada
T05	Inferencia texto CPU	Generación y tok/s	Verificada: 10,4 tok/s
T06	Inferencia texto RPC	Generación y trazas RPC0	Verificada: 8,6 tok/s y capas RPC0
T07	Webcam + YOLO	Video y captura JPEG reales	Confirmada por usuario, con lag
T08	Qwen3-VL imagen subida	Descripción real y tiempo	No existe captura de resultado final verificada
T09	Fotograma YOLO → Qwen3-VL	Respuesta de captura original	Falló en 2.0.2; 2.0.3 pendiente
T10	Apagar Worker	Nueva respuesta válida en CPU VM	Pendiente prueba completa
T11	Offline	Inferencia sin conectividad WAN	Pendiente
T12	Persistencia IP	IP estable luego de reiniciar	Pendiente
T13	iperf3	Medición Mbit/s ambos sentidos	Pendiente

17.2 Registro de calidad de visión (plantilla editable)

Muestra	Objeto / expectativa	Salida YOLO	Salida Qwen3-VL	Segundos	Juicio
01	[Fotografía propia de etiqueta]	[Clase/confianza]	[Descripción literal]	[]	[]

Muestra	Objeto / expectativa	Salida YOLO	Salida Qwen3-VL	Segundos	Juicio
02	[Cámara: botella/caja]	[Clase/confianza]	[Descripción literal]	[]	[]
03	[Escena con 2 objetos]	[Clases/confianza]	[Descripción literal]	[]	[]

Se recomienda evaluar precisión sin inventar porcentajes: número de objetos reales detectados correctamente, falsas detecciones y aciertos semánticos del VLM en al menos tres fotografías. Un detector COCO preentrenado no reconoce automáticamente códigos SKU específicos ni texto diminuto con fiabilidad; documentar estos límites.

17.3 Control de evidencias de consola

```
# Crear evidencia en VM:
mkdir -p ~/edge-vision-lab/evidencias
bash ~/edge-vision-lab/scripts/recoger_evidencias.sh
# Extracto de asignación CPU/RPC:
grep -Ei 'RPC0|offload|assigned to device|compute buffer' ~/edge-vision-lab/evidencias/modelo-vlm.log |
tail -n 40
# Monitoreo:
htop
# Worker durante inferencia por SSH:
ssh orangepi@192.168.100.20 htop
```

18. Análisis económico, seguridad, limitaciones y riesgos

18.1 Justificación financiera de IA local

Una API externa de IA suele cobrar por solicitudes, tokens y/o imágenes, además de requerir disponibilidad de red. EdgeVision demuestra la viabilidad de mantener el modelo, las fotografías y el procesamiento dentro de recursos propios. **No se calcularon costos unitarios ni un ROI numérico**, porque faltan mediciones de consumo eléctrico, valor de amortización, número de consultas y costo real de la alternativa SaaS. No se debe afirmar “cero costo”; el ahorro relevante sería la eliminación del cargo variable por tokens externos durante inferencia.

18.2 Controles mínimos de seguridad

- Segmentar servicios internos (RPC 50052, cámara 9200 y métricas 9100) y no publicarlos en router ni Internet.
- La aplicación FastAPI y RPC no implementan autenticación/HTTPS de producción; usarlos solo en laboratorio LAN controlado.
- Establecer contraseñas seguras y, como mejora, llaves SSH sin contraseña y firewall por origen IP.
- Limitar entradas de imágenes a formatos y tamaños definidos; deshabilitar subida de contenido privado y vigilar registros.
- Corregir hora/NTP para validez de descargas, logs y firmas; guardar respaldo de configuraciones y versiones del modelo.

- Observar restricciones de licencias de modelos y dependencias antes de un despliegue comercial, no asumir derechos implícitos.

18.3 Matriz de riesgos del laboratorio

Riesgo	Impacto	Mitigación y evidencia
RAM insuficiente VLM	Carga aborta / OOM	Una instancia, modelo 2B Q4, monitor y contexto acotado.
Worker RPC desconectado	Retraso en inferencia	Watchdog y failover local en frío; prueba repetible.
Wi-Fi compartido / IP temporal	Pérdida de SSH al reiniciar	Direcciones persistentes luego de validar perfiles y `netplan try`.
Fallo de enlace/switch	Clúster inaccesible	Único switch es punto común de falla; documentarlo.
Retraso webcam CPU	Pocos FPS	1 FPS objetivo, threads 2, frame reducido, pausa en carga VLM.
Error formato ONNX	No inicia YOLO	Modelo rectangular 480×640 y validación del hash.
Descripción inventada	Respuesta semántica incorrecta	Prompt explícito, contrastar con fotografía y medir error.
Reinicio VLM durante captura	Fallo temporal de análisis	Parche 2.0.3 con recuperación local y reintento.

18.4 Gobernanza y normativa

La discusión académica aborda privacidad, gobierno de datos, plataformas de IA como servicio, gasto recurrente frente a infraestructura propia y continuidad operativa. No se ha determinado que este laboratorio requiera una licencia específica de telecomunicaciones ni que la Ley General de Telecomunicaciones regule directamente el modelo usado; cualquier conclusión jurídica debe basarse en análisis normativo verificable. El informe deja constancia de buenas prácticas técnicas: no exponer puertos, acceso limitado y no transferir imágenes a una API comercial.

19. Matriz de cumplimiento: requisitos y rúbrica oficial

19.1 Requisitos escritos y explicados en clase

Exigencia	Solución EdgeVision	Evaluación de cumplimiento
Placa física de bajo consumo	Orange Pi 4 Pro ARM64, Debian 12	Sí, evidencia captura del SO.
Segundo equipo en red	Laptop Win11 con VM Ubuntu x86-64	Sí en clase; compatibilidad PDF verbal, no confirmada por escrito.
Switch Gigabit y CAT6	Interconexión directa por switch 5 puertos	Sí, Ping/SSH y enlace 1000Mb/s; throughput pendiente.

Exigencia	Solución EdgeVision	Evaluación de cumplimiento
Subnet privada, IP estáticas	Plan 192.168.100.0/24 con .10/.20	En uso temporal; persistencia pendiente.
Procesamiento local sin IAaaS	Pesos GGUF/ONNX y API local	Arquitectura satisface diseño; offline formal pendiente.
Unir las CPU mediante middleware	ggml RPC con capas RPC0	Evidencia textual; VLM distribuido requiere logs propios.
Visión mediante webcam YOLO/MobileNet	Webcam USB + YOLO11-Nano ONNX ARM64	Usuario confirmó video y captura; medir FPS.
Subir imagen a LLM multimodal	Qwen3-VL + portal de imagen	Portal y captura presentes; respuesta real por cámara pendiente.
Caída Worker sin perder capacidad	Controlador de failover frío CPU VM	Implementado; prueba completa pendiente.
Informe técnico detallado	Este documento, comandos y anexos	Word generado; exportar a PDF para Canvas.
Auto/coevaluación y videos	Procedimientos y plantillas de anexos	Producir material real, no inventar integrantes/nota.
Feria virtual	Informe, video principal y reflexión	Publicación y acceso pendientes.

19.2 Rúbrica PDF: ocho criterios y evidencia asociada

Los encabezados y máximos consignados en R1 se enumeran abajo. Algunos textos descriptivos de la rúbrica contienen términos ajenos al proyecto de IA (pfSense, WAN dual stack, portal cautivo, telecomunicaciones) y se observan parcialmente truncados en el PDF; la explicación del catedrático orienta su valoración hacia el caso de uso Edge Computing.

Criterio de rúbrica	Máx.	Dónde se demuestra en el informe/proyecto
Estructura e integración del contenido	2	Problema, objetivos, arquitectura, caso de visión: §§1–7.
Profundidad técnica y análisis académico	2	CPU heterogénea, RPC, GGUF/Q4, fallas, benchmarking: §§4, 10–18.
Propuesta final del proyecto colaborativo	2	Roles reales y dominio de los nodos por todos: §20.
Autoevaluación — Anexo A	2	Hoja oficial de Canvas con puntuación 1–100 por integrante.
Valoración de pares — Anexo B	2	Hoja de coevaluación cruzada con puntuación 1–100.
Defensa técnica individual en feria	6	Videos/preguntas técnicas y demostración guiada: §20.
Presentación formal del informe	2	Documento Word estructurado y PDF de entrega.
Cumplimiento técnico y evidencias	2	Capturas, logs, pruebas y matriz T01–T13: §§6–19.

Total del esquema mostrado: 20 puntos. La distribución de fases (10 y 10 puntos) también aparece en la primera página del PDF. Las calificaciones definitivas las determina el catedrático, no este informe.

20. Entregables obligatorios, trabajo en equipo y defensa

20.1 Paquete de entrega: aplicar la instrucción más completa

Material	Documento escrito (R1)	Clase oral (R2)	Acción operativa
Informe	Un PDF formal grupal por coordinador	Amplio, técnico, casos, pruebas, recomendaciones, conclusiones	Exportar este Word a PDF; comprobar legibilidad y compartir acceso.
Video individual	Un enlace por estudiante, 5–7 min	Todos deben dominar respuestas técnicas	Grabar enlace individual; Canvas de cada miembro.
Video de demostración principal	No descrito en p.1	Hasta ~10 min, completo; feria virtual	Video grupal con ejecución real y CPU/RPC en pantalla.
Video de reflexión	No descrito en p.1	Participantes explican aprendizajes/límites	Grabar comentarios genuinos de participantes.
Presentación ante catedrático	Fase de defensa	Aproximadamente 5 min; puede preguntar a cualquiera	Ensayar demostración en vivo y preguntas.
Auto/coevaluación	Parte de fase individual	Excel de Canvas con escala 1–100	Completar plantilla oficial, sin asignar notas ficticias.
Publicación feria virtual	No precisado en p.1	Página común coordinada por el equipo web	Publicar videos, informe y reflexión; probar permisos.
Canvas	PDF y videos individuales	Todos los integrantes entregan viernes 16	Cada miembro verificar envío y hora límite en Canvas.

20.2 Contenido obligatorio del informe de acuerdo con la clase

- **Estructura del problema:** organización no desea gastar tokens en IAaaS y necesita privacidad/autonomía.
- **Hardware:** justificar Orange Pi, VM, CPUs heterogéneas, switch, CAT6 y cámara.
- **Modelo:** justificar Qwen3-VL + cuantización Q4, mmproj y YOLO-Nano ONNX; distinguir alcance de ambos.
- **Procedimientos:** instalación reproducible, interfaces, puertos, API, scripts, lógica de conmutación.
- **Evaluación:** desempeño CPU/RAM, tasas reales, exactitud visual y errores observados.

- **Conclusiones y recomendaciones:** incluir límites, costo eléctrico, privacidad, SPOF y operación sin Internet.

20.3 Roles y evidencia de participación (completar con nombres reales)

Integrante	Responsabilidad asignada	Evidencia de participación
[Nombre A]	Placa Orange Pi: imagen Debian, USB, YOLO y monitoreo	[Captura/bitácora propia]
[Nombre B]	VirtualBox, Ubuntu, red Ethernet/ICS/SSH e IPs	[Captura/bitácora propia]
[Nombre C]	Compilación llama.cpp, RPC, modelos GGUF, benchmark	[Captura/bitácora propia]
[Nombre D]	Portal web, captura, failover, pruebas e informe	[Captura/bitácora propia]

Esta distribución es una **plantilla**, no afirmación de que esas personas hayan realizado las tareas. Registrar roles, aportes, videos y notas reales; todos deben poder explicar el diseño completo.

20.4 Guion de demostración grupal (hasta ~10 minutos)

Minuto	Actividad	Pantalla / evidencia
00:00–01:00	Problema de negocio y objetivo local autónomo	Diagrama e inventario físico
01:00–02:15	Topología ARM64 ↔ x86-64 + switch Gigabit	Foto del montaje, IPs y SSH
02:15–03:30	Q4, GGUF, mmproj y qué significa RPC	Logs `RPC0`/capas/buffers
03:30–05:10	YOLO en webcam; recuadros, clases, FPS	Portal cámara + objeto real
05:10–07:00	Captura de fotograma y análisis Qwen3-VL	Respuesta real y duración
07:00–08:30	Detener Worker y demostrar respaldo (o video registrado)	Pantalla modo Solo VM; comparar
08:30–09:30	Prueba sin Internet, métricas, límites y costo	WAN desconectada, LAN operativa
09:30–10:00	Conclusiones y lecciones del equipo	Cierre y rutas a evidencias

20.5 Defensa individual de 5–7 minutos y preguntas frecuentes

Cada integrante debe explicar sin leer: problema de negocio, CPU local versus cloud, características Orange Pi, bridged/NAT/ICS, diferencias YOLO vs VLM, papel de GGUF/mmproj, qué muestra RPC0, por qué 8,6 tok/s pudo ser menor que 10,4, funcionamiento de watchdog, límites del failover, prueba sin Internet y seguridad del laboratorio.

- ¿Se juntan realmente los núcleos de Linux? — No. RPC distribuye partes del grafo de cálculo entre dispositivos; no convierte dos kernels en un sistema SMP.
- ¿Por qué la VM puede marcar 100 % y la Orange Pi 25 %? — La carga depende del reparto y la ruta activa; en modo Solo VM no demuestra cálculo remoto.

- ¿Qué pasa si se apaga la Orange Pi? — El VLM puede recargarse en CPU VM para nuevas solicitudes, con demora; cámara YOLO queda indisponible.
- ¿Qué ocurre sin Internet? — Una vez descargados los modelos, CPU y datos pueden trabajar por LAN; falta dejar evidencia de prueba WAN caída.
- ¿Cuál es el costo? — Ya no se cobra token por inferencia de terceros, pero existen costo de hardware, energía, operación y mantenimiento.
- ¿Por qué hay dos modelos? — YOLO detecta/localiza objetos por fotograma y Qwen3-VL interpreta una escena; ambos apoyan el mismo caso de visión.

20.6 Excel de evaluaciones: pautas sin inventar resultados

El catedrático indicó que cada integrante anote una **autoevaluación de 1 a 100**, y que se realice **valoración por pares de 1 a 100**, asignando personas distintas. Debe utilizarse el Excel oficial adjunto en Canvas. No se incluyó esa plantilla en la conversación, por lo que este informe no la sustituye. Completar nombres, notas y justificación con participación real; no asignar puntuaciones predeterminadas ni ficticias.

21. Conclusiones y recomendaciones

- Se implementó una infraestructura real ARM64/x86-64 con Linux en ambos nodos, interconectados por switch Gigabit; Ping/SSH y enlace negociado a 1000 Mb/s fueron comprobados.
- La misma revisión de `llama.cpp` fue compilada en ambas arquitecturas y permitió inferencia textual con asignación de recursos RPC0. Un modelo pequeño no se aceleró: pasó de 10,4 a 8,6 tok/s, resultado coherente con overhead de distribución.
- EdgeVision amplió el caso de visión incorporando YOLO11-Nano con webcam USB físicamente conectada a Orange Pi y Qwen3-VL para análisis semántico de fotografías. El usuario confirmó que la webcam mostró imagen y capturó un fotograma, con cierto retraso.
- Las incidencias encontradas (puente Wi-Fi erróneo, objetivo CMake ausente, reloj, SHA del ONNX, tamaño rectangular del tensor, VLM no listo) se analizaron y resolvieron o abordaron con parches de alcance limitado.
- La persistencia de IP, la respuesta real de la captura tras el parche 2.0.3, el failover extremo a extremo, iperf3 y el experimento offline deben **verificarse antes de declararlos finalizados**.
- El proyecto ofrece una prueba convincente de viabilidad de IA local con hardware común, pero requiere ajustes de seguridad, capacidad, observabilidad y tolerancia a puntos únicos de falla si se traslada a producción.

21.1 Acciones prioritarias para cerrar la entrega

- 1. Encender laboratorio y validar el parche 2.0.3: captura JPEG → respuesta Qwen3-VL; guardar pantalla/log.
- 2. Confirmar logs RPC del VLM, o declarar demostración distribuida únicamente con modelo Qwen2.5 de texto.

- 3. Aplicar IP persistentes con pruebas después de reinicio y registrar `iperf3`.
- 4. Probar corte de Worker y nueva consulta en CPU VM; registrar demora y resultado.
- 5. Ejecutar prueba sin Internet con WAN realmente aislada.
- 6. Exportar Word a PDF, incorporar fotos reales de montaje y respuestas definitivas, grabar videos, completar Excel de pares y publicar en feria/Canvas.

A. Anexo A — Guía rápida de comandos por sistema

A.1 Windows 11 PowerShell

```
Get-NetAdapter | Format-Table Name, InterfaceDescription, Status, LinkSpeed
Get-NetIPAddress -AddressFamily IPv4 |
    Format-Table InterfaceAlias, IPAddress, PrefixLength
Test-NetConnection 192.168.137.128 -Port 22
ssh milo@192.168.137.128
ssh -p 2222 milo@127.0.0.1
# Lanzadores después de instalación:
# Iniciar-EdgeVision.bat
# Reparar-Integracion-Qwen-203.bat (aplicar una sola vez)
# Diagnosticar-EdgeVision-203.bat
```

A.2 Ubuntu: red, modelo, estado y logs

```
ip -4 -br addr
ip route
ping -c 4 192.168.100.20
nc -vz -w 3 192.168.100.20 50052
ssh orange@192.168.100.20
ls -lh ~/edge-vision-models/
bash ~/edge-vision-lab/scripts/start.sh
bash ~/edge-vision-lab/scripts/status.sh
bash ~/edge-vision-lab/scripts/recoger_evidencias.sh
tail -n 80 ~/edge-vision-lab/evidencias/modelo-vlm.log
bash ~/edge-vision-lab/scripts/stop.sh
```

A.3 Orange Pi: red, USB, procesos y logs

```
uname -m; free -h; nproc; ip -4 -br addr
sudo ethtool end0 | grep -E 'Speed:|Duplex:|Link detected:'
tmux ls
v4l2-ctl --list-devices
ls -l /dev/video*
curl --noproxy '*' http://192.168.100.20:9200/status
tail -n 80 ~/edgevision-camera/camara.log
htop
# Sesiones tmux típicas: edge-rpc y edgevision-camera
```

B. Anexo B — Registro de evidencias visuales del laboratorio

Las capturas siguientes son pruebas de laboratorio proporcionadas durante la construcción. No se presentan como mediciones obtenidas a posteriori ni sustituyen las demostraciones finales pendientes. Las más recientes, correspondientes a respuestas multimodales y video en vivo, deberán incorporarse al completar la validación.

```
--
-- = Pass remaining options to the native tool.
milo@server:~/llama.cpp$ cmake --build build --parallel 3 --target llama-server llama-cli ggml-rpc-server
[1014] error: unknown target 'ggml-rpc-server'
milo@server:~/llama.cpp$
milo@server:~/llama.cpp$ cmake --build build --parallel 3 --target llama-server llama-cli
[146/228] Provisioning UI assets
-- UI: downloading from b11626: https://huggingface.co/buckets/ggml-org/llama-ui/resolve/b11626/dist.tar.gz
-- UI: download dist.tar.gz.sha256 from b11626 failed: "HTTP response code said error"
-- UI: downloading from latest: https://huggingface.co/buckets/ggml-org/llama-ui/resolve/latest/dist.tar.gz
-- UI: archive verified and extracted
-- UI: HF download succeeded, stamp updated (ggml-org/llama-ui/b11626, resolved: latest)
-- UI: gzip compression applied (/home/milo/llama.cpp/build/tools/ui/ui-gzip_gzip)
-- UI: embedded 70 assets
[223/228] Linking CXX executable bin/llama-cli
milo@server:~/llama.cpp$ grep -E "(GGML_RPC|LLAMA_BUILD_TOOLS):" build/CMakeCache.txt
GGML_RPC:BOOL=OFF
LLAMA_BUILD_TOOLS:BOOL=ON
milo@server:~/llama.cpp$ cmake -S . -B build -DGGML_RPC=ON -DGGML_NATIVE= -DLLAMA_BUILD_TOOLS=ON
-- llama.cpp version: 0.6.0-dev
-- CMAKE_BUILD_TYPE=Release
-- ccache found, compilation results will be cached. Disable with GGML_CCACHE=OFF.
-- CMAKE_SYSTEM_PROCESSOR: x86_64
-- GGML_SYSTEM_ARCH: x86
-- Including CPU backend
-- X86 detected
-- Adding CPU backend variant ggml-cpu:
-- Using RPC backend
-- RDMA transport disabled
-- Including RPC backend
-- ggml version: 0.26.0
-- ggml commit: df18bee11
-- OpenSSL found: 3.0.13
-- Generating embedded license file for target: llama-app
-- Configuring done (0.7s)
-- Generating done (0.2s)
-- Build files have been written to: /home/milo/llama.cpp/build
milo@server:~/llama.cpp$ grep -E "(GGML_RPC|LLAMA_BUILD_TOOLS):" build/CMakeCache.txt
GGML_RPC:BOOL=ON
LLAMA_BUILD_TOOLS:BOOL=ON
milo@server:~/llama.cpp$ cmake --build build --parallel 3 --target ggml-rpc-server
[20/20] Linking CXX executable bin/ggml-rpc-server
milo@server:~/llama.cpp$ ls -lh build/bin/ggml-rpc-server
-rwxrwxr-x 1 milo milo 193K Oct 14 07:36 build/bin/ggml-rpc-server
milo@server:~/llama.cpp$ ls -lh build/bin/llama-server build/bin/llama-cli build/bin/ggml-rpc-server
-rwxrwxr-x 1 milo milo 1.4M Oct 14 07:30 build/bin/ggml-rpc-server
-rwxrwxr-x 1 milo milo 1.4M Oct 14 07:30 build/bin/llama-cli
-rwxrwxr-x 1 milo milo 18K Oct 14 07:30 build/bin/llama-server
milo@server:~/llama.cpp$ git rev-parse HEAD
410bee11e96572607ac3b1d0976f3c7ac3165
milo@server:~/llama.cpp$
```

Evidencia B-1. Compilación y presencia de los binarios llama-server, llama-cli y ggml-rpc-server en Ubuntu.

```
EdgeVision-Master [Corriendo] - Oracle VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
- ARM detected flags: -mcpu=cortex-a76.cortex-a55+crypto
- Performing Test GGML_MACHINE_SUPPORTS_dotprod - Success
- Performing Test GGML_MACHINE_SUPPORTS_18mm - Success
- Performing Test GGML_MACHINE_SUPPORTS_18mm - Failed
- Performing Test GGML_MACHINE_SUPPORTS_no18mm - Success
- Performing Test GGML_MACHINE_SUPPORTS_no18mm - Success
- Performing Test GGML_MACHINE_SUPPORTS_sve - Success
- Performing Test GGML_MACHINE_SUPPORTS_sve - Failed
- Performing Test GGML_MACHINE_SUPPORTS_nosve - Success
- Performing Test GGML_MACHINE_SUPPORTS_nosve - Success
- Performing Test GGML_MACHINE_SUPPORTS_sme - Failed
- Performing Test GGML_MACHINE_SUPPORTS_sme - Failed
- Performing Test GGML_MACHINE_SUPPORTS_nosme - Success
- Performing Test GGML_MACHINE_SUPPORTS_nosme - Failed
- Checking for ARM features using flags:
- -U_ARM_FEATURE_MATHUL_INT8
- -U_ARM_FEATURE_SVE
- -mcpu=cortex-a76.cortex-a55+crypto+dotprod+no18mm+nosve
- Performing Test HAVE_DOTPROD - Success
- Performing Test HAVE_DOTPROD - Success
- Performing Test HAVE_SVE - Success
- Performing Test HAVE_SVE - Failed
- Performing Test HAVE_MATHUL_INT8 - Failed
- Performing Test HAVE_MATHUL_INT8 - Failed
- Performing Test HAVE_FMA - Success
- Performing Test HAVE_FMA - Success
- Performing Test HAVE_FP16_VECTOR_ARITHMETIC - Success
- Performing Test HAVE_SME - Success
- Performing Test HAVE_SME - Failed
- Adding CPU backend variant ggml-cpu: -U_ARM_FEATURE_MATHUL_INT8;-U_ARM_FEATURE_SVE;-mcpu=cortex-a76.cortex-a55+crypto+dotprod
-- Using RPC backend
-- RDMA transport disabled
-- Including RPC backend
-- ggml version: 0.26.0
-- ggml commit: df18bee11
-- Found OpenSSL: /usr/lib/aarch64-linux-gnu/libcrypto.so (found version "3.0.20")
- Performing Test OPENSSL_VERSION_SUPPORTED - Success
- Performing Test OPENSSL_VERSION_SUPPORTED - Success
- OpenSSL found: 3.0.20
-- Generating embedded license file for target: llama-app
-- Configuring done
-- Generating done
-- Build files have been written to: /home/orangepi/llama.cpp/build
orangepi@orangepi4pro:~/llama.cpp$ cmake --build build --parallel 2 --target ggml-rpc-server
[1/41] Linking CXX executable bin/ggml-rpc-server
orangepi@orangepi4pro:~/llama.cpp$ ls -lh build/bin/ggml-rpc-server
-rwxr-xr-x 1 orangepi orangepi 173K Oct 14 09:04 build/bin/ggml-rpc-server
orangepi@orangepi4pro:~/llama.cpp$
```

Evidencia B-2. Compilación de ggml-rpc-server sobre ARM64 en Orange Pi.

```

Connection to 192.168.137.20 closed.
milo@server:~$ ping -c 192.168.100.20
ping: invalid argument: '192.168.100.20'
milo@server:~$ ping -c 4 192.168.100.20
PING 192.168.100.20 (192.168.100.20) 56(84) bytes of data.

--- 192.168.100.20 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3089ms

milo@server:~$ ip -4 -br addr
lo                UNKNOWN      127.0.0.1/8
enp0s3            UP            192.168.137.128/24 metric 100
enp0s8            UP            10.0.3.15/24 metric 100
milo@server:~$ sudo ip addr add 192.168.100.10/24 dev enp0s3
[sudo] password for milo:
milo@server:~$ ip route get 192.168.100.20
192.168.100.20 dev enp0s3 src 192.168.100.10 uid 1000
    cache
milo@server:~$ ping -I 192.168.100.10 -c 4 192.168.100.20
PING 192.168.100.20 (192.168.100.20) from 192.168.100.10 : 56(84) bytes of data.
64 bytes from 192.168.100.20: icmp_seq=1 ttl=64 time=2.32 ms
64 bytes from 192.168.100.20: icmp_seq=2 ttl=64 time=1.52 ms
64 bytes from 192.168.100.20: icmp_seq=3 ttl=64 time=2.29 ms
64 bytes from 192.168.100.20: icmp_seq=4 ttl=64 time=2.09 ms

--- 192.168.100.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.519/2.054/2.323/0.322 ms
milo@server:~$ ip -4 -br addr
lo                UNKNOWN      127.0.0.1/8
enp0s3            UP            192.168.137.128/24 metric 100 192.168.100.10/24
enp0s8            UP            10.0.3.15/24 metric 100
milo@server:~$

```

Evidencia B-3. Ping de Ubuntu a la Orange Pi: 4 transmitidos, 4 recibidos, 0 % de pérdida.

```

0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 20 no actualizados.
milo@server:~$ nc -vz -w 3 192.168.100.20 50052
Connection to 192.168.100.20 50052 port [tcp/*] succeeded!
milo@server:~$ mkdir -p ~/models
milo@server:~$ curl -fL --retry 3 -o ~/models/quen2.5-0.5b-instruct-q4_k_m.gguf "https://huggingface.co/Quen/Quen2.5-0.5B-Instruct-GGUF/resolve/main/quen2.5-0.5b-instruct-q4_k_m.gguf"
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 1120  100 1120    0     0   5194      0 --:--:-- --:--:-- --:--:--  5161
100 469M  100 469M    0     0  11.6M      0 0:00:40 0:00:40 --:--:-- 10.3M
milo@server:~$ ls -lh ~/models/
total 469M
-rw-r--r-- 1 milo milo 469M oct 14 09:35 quen2.5-0.5b-instruct-q4_k_m.gguf
milo@server:~$ cd ~/llama.cpp
milo@server:~/llama.cpp$ ./build/bin/llama-cli -m ~/models/quen2.5-0.5b-instruct-q4_k_m.gguf -p "Explica que es Edge Computing en tres frases" -n 80 -c 1024 -t 3 -ngl 0

Loading model...

llama.cpp

build      : b11626-df18bee11
model      : /home/milo/models/quen2.5-0.5b-instruct-q4_k_m.gguf
ftype     : Q4_K - Medium
modalities : text

available commands:
./exit or Ctrl+C    stop or exit
./regen             regenerate the last response
./clear             clear the chat history
./read <file>       add a text file
./glob <pattern>    add text files using globbing pattern

> Explica que es Edge Computing en tres frases
Edge Computing es una técnica de tecnologías de computación que se utiliza para reducir el tiempo de procesamiento y almacenamiento de datos y para optimizar la entrega y entrega de datos a la parte de la red más cercana a la cliente, es decir, a los dispositivos personales. En otras palabras, la tecnología Edge Computing se enfoca en procesar datos en la parte de

[ Prompt: 12,0 t/s | Generation: 10,4 t/s ]

```

Evidencia B-4. Generación de texto local con Qwen2.5-0.5B (~10,4 tokens/s).

```

Exiting...
milo@server:~/llama.cpp$ ./build/bin/llama-cli -m ~/models/qwen2.5-0.5b-instruct-q4_k_m.gguf --rpc 192.168.100.20:50052 -ngl 8 -t 3 -c 1024 -n 80 -p "Explica
es Edge Computing en tres frases"

Loading model...

llama.cpp

build      : b11626-df18bee11
model      : /home/milo/models/qwen2.5-0.5b-instruct-q4_k_m.gguf
ftype      : Q4_K - Medium
modalities : text

available commands:
  /exit or Ctrl+C    stop or exit
  /regen             regenerate the last response
  /clear             clear the chat history
  /read <file>       add a text file
  /glob <pattern>    add text files using globbing pattern

> Explica que es Edge Computing en tres frases
Edge Computing es una técnica de red de computación que se utiliza para recopilar datos en la parte física de la red de una empresa o organización, y enviarlo
a una base de datos en la red de red. Esta técnica reduce el tamaño y el tiempo de almacenamiento necesarios para procesar y analizar los datos, lo que aument
la velocidad y eficiencia del procesamiento

[ Prompt: 14,7 t/s | Generation: 8,6 t/s ]

>

```

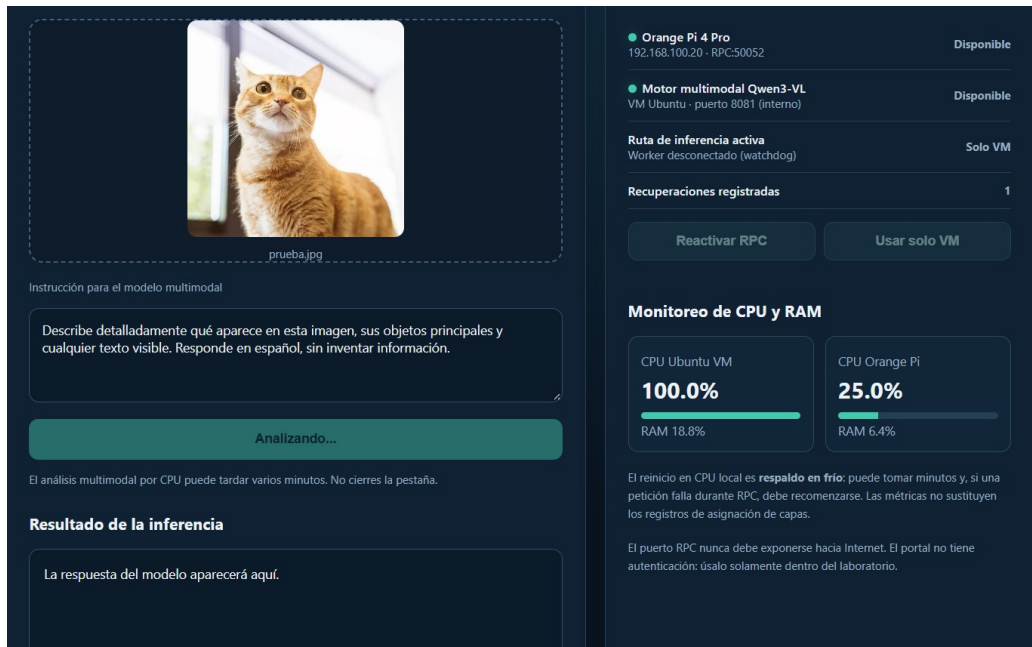
Evidencia B-5. Generación de texto con `--rpc` (~8,6 tokens/s).

```

0.00.480.798 D llama_kv_cache: layer 8: dev = CPU
0.00.480.799 D llama_kv_cache: layer 9: dev = CPU
0.00.480.801 D llama_kv_cache: layer 10: dev = CPU
0.00.480.802 D llama_kv_cache: layer 11: dev = CPU
0.00.480.803 D llama_kv_cache: layer 12: dev = CPU
0.00.480.803 D llama_kv_cache: layer 13: dev = CPU
0.00.480.805 D llama_kv_cache: layer 14: dev = CPU
0.00.480.806 D llama_kv_cache: layer 15: dev = CPU
0.00.480.807 D llama_kv_cache: layer 16: dev = CPU
0.00.480.810 D llama_kv_cache: layer 17: dev = RPC0
0.00.480.813 D llama_kv_cache: layer 18: dev = RPC0
0.00.480.814 D llama_kv_cache: layer 19: dev = RPC0
0.00.480.816 D llama_kv_cache: layer 20: dev = RPC0
0.00.480.817 D llama_kv_cache: layer 21: dev = RPC0
0.00.480.818 D llama_kv_cache: layer 22: dev = RPC0
0.00.480.819 D llama_kv_cache: layer 23: dev = RPC0
0.00.480.822 I llama_kv_cache: CPU KV buffer size = 0,00 MiB
0.00.480.824 I llama_kv_cache: RPC0[192.168.100.20:50052] KV buffer size = 0,00 MiB
0.00.480.829 I llama_kv_cache: size = 12,00 MiB ( 1024 cells, 24 layers, 1/1 seqs), K (f16): 6,00 MiB, V (f16): 6,00 MiB
0.00.544.378 I sched_reserve: RPC0[192.168.100.20:50052] compute buffer size = 33,01 MiB
0.00.544.379 I sched_reserve: CPU compute buffer size = 38,27 MiB
0.00.550.756 I common_memory_breakdown_print: | memory breakdown [MiB] | total free self mo
del context compute unaccounted |
0.00.550.846 I common_memory_breakdown_print: | - RPC0 (192.168.100.20:50052) | 3844 = 3844 + ( 241 =
205 + 3 + 33) + -241 |
0.00.650.349 I llama_model_loader: - kv 13: qwen2.attention.layer_norm_rms_epsilon f32 =
0,000001
0.00.728.025 I llama_prepare_model_devices: using device RPC0 (192.168.100.20:50052) (unknown id) - 3844 MiB free
0.01.106.819 I print_info: n_layer = 24
0.01.106.820 I print_info: n_layer_all = 24
0.01.131.905 D load_tensors: layer 0 assigned to device CPU, is_swa = 0
0.01.131.906 D load_tensors: layer 1 assigned to device CPU, is_swa = 0
0.01.131.907 D load_tensors: layer 2 assigned to device CPU, is_swa = 0
0.01.131.907 D load_tensors: layer 3 assigned to device CPU, is_swa = 0
0.01.131.908 D load_tensors: layer 4 assigned to device CPU, is_swa = 0
0.01.131.908 D load_tensors: layer 5 assigned to device CPU, is_swa = 0
milo@server:~/llama.cpp$

```

Evidencia B-6. Capas del modelo y buffers de cómputo asignados a RPC0 y CPU.



Evidencia B-7. Interfaz Qwen3-VL y tablero de recursos: captura durante análisis, no respuesta final.

C. Anexo C — Estructura del proyecto y configuración

```
~/llama.cpp/
  build/bin/llama-server
  build/bin/llama-cli
  build/bin/ggml-rpc-server
~/edge-vision-models/
  Qwen3VL-2B-Instruct-Q4_K_M.gguf
  mmproj-Qwen3VL-2B-Instruct-Q8_0.gguf
~/edge-vision-lab/
  backend/engine.py
  backend/main.py
  backend/static/index.html
  backend/.env
  scripts/{start.sh,stop.sh,status.sh,install_vm.sh,...}
  evidencias/{portal.log,modelo-vm.log,inferencias.jsonl,...}
~/edgevision-camera/ [EN ORANGE PI]
  camera_service.py
  yolo11n.onnx
  start.sh
  camara.log
  .venv/
```

Configuración documentada del portal: VM_THREADS=3; RPC_LAYERS=8;
 CONTEXT_TOKENS=4096; IMAGE_MAX_TOKENS=512; EDGE_UPSTREAM_TIMEOUT=480;
 EDGE_BOOT_TIMEOUT=210; EDGE_CAMERA_URL=http://192.168.100.20:9200;
 EDGE_RPC_HOST=192.168.100.20; EDGE_RPC_PORT=50052.

D. Anexo D — Registro de pruebas para completar

Código	Fecha	Evidencia / ruta	Responsabl e	Resultado
T08	[]	[Captura de respuesta final Qwen3-VL]	[]	[]
T09	[]	[Fotograma webcam → respuesta tras 2.0.3]	[]	[]
T10	[]	[Salida `vm-fallback` y nueva respuesta]	[]	[]
T11	[]	[WAN caída + LAN operativa + respuesta]	[]	[]
T12	[]	[Dos IP tras reiniciar / SSH estable]	[]	[]
T13	[]	[iperf3 ida y vuelta]	[]	[]
EV-A	[]	[Autoevaluación Canvas anexo oficial]	[]	[]
EV-B	[]	[Coevaluación Canvas anexo oficial]	[]	[]
V-G	[]	[Link video grupal feria]	[]	[]
V-I	[]	[Links videos individuales]	[]	[]
V-R	[]	[Link video reflexión]	[]	[]

E. Anexo E — Lista de entrega y permisos de acceso

- [] Portada e integrantes con nombres/carnés verificados.
- [] Informe DOCX revisado, exportado a PDF final para Canvas.
- [] Fotos de montaje con laptop, switch Gigabit, Orange Pi, cables y webcam.
- [] Logs del RPC y prueba de Qwen2.5; captura real de cámara YOLO y del VLM.
- [] Medición final CPU/RAM/FPS e interpretación correcta de fotografía.
- [] Resultado probado de recuperación de Worker y ejecución offline.
- [] Video grupal de demostración y publicación en la feria virtual.
- [] Video individual de 5–7 min por estudiante y video de reflexión.
- [] Hoja oficial de autoevaluación y valoración por pares con datos auténticos.
- [] Cada integrante confirma su entrega individual de Canvas.
- [] Todos los enlaces se abren sin pedir permisos adicionales al evaluador.
- [] Equipo ensayó preguntas técnicas y demo de 5 min para el 17 de octubre.

F. Anexo F — Fuentes y referencias consultadas

[R1] Proyecto final requerimientos.pdf, 8 páginas, material entregado para el curso. Título del proyecto, restricciones técnicas, casos de uso y rúbrica oficial.

[R2] class.txt, transcripción de la clase de Arquitectura de Computadoras del 10 de octubre de 2026. Instrucciones orales sobre hardware, VM, RPC, Canvas, feria, videos y rúbrica.

[R3] Capturas, resultados de terminal y confirmaciones del laboratorio EdgeVision compartidas entre el 13 y 15 de octubre de 2026.

[R4] Código de los paquetes EdgeVision: proyecto inicial; perfil Win11/OrangePi 4 GB; Fase Final Multimodal; actualización Cámara/YOLO 2.0; parches 2.0.1, 2.0.2 y 2.0.3.

[R5] Proyecto ggml-org / llama.cpp, repositorio de referencia:
<https://github.com/ggml-org/llama.cpp> ; revisión compilada
df18bee11e96557b07aa7b12db0976f3c7ac9165.

[R6] Pesos GGUF Qwen3-VL-2B-Instruct: <https://huggingface.co/Qwen/Qwen3-VL-2B-Instruct-GGUF> .

[R7] Pesos Qwen2.5-0.5B para prueba: <https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct-GGUF> .

[R8] Exportación YOLO11n ONNX utilizada en el kit:
<https://huggingface.co/unileon-robotics/YOLO11-ONNX> .

[R9] Documentación de Ubuntu Server y VirtualBox para instalación de VM, SSH y redes:
<https://ubuntu.com/server/docs> ; <https://www.virtualbox.org/manual/> .

[R10] Manual/descargas del fabricante Orange Pi 4 Pro: <https://www.orangepi.org/> .