

UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA
FACULTAD DE INGENIERÍA EN SISTEMAS
CATEDRÁTICO: ING. Alex Garcia

***Proyecto Final Integrador: diseño, montaje e implementación
de un clúster de edge computing para inferencia de IA local
autónoma***

Geovanny Alexander Maldonado Barraza	0900-23-25502
Hever Rafael Marroquin Garcia	0900-16-625
Mary Denisse Navas González	0900-23-1303
Edwin Robert Cáceres Urizar	0900-11-8036

16 de Octubre, 2026

Índice

Índice.....	2
Resumen ejecutivo.....	4
1. Introducción.....	5
2. Definición del problema.....	6
2.1 Contexto.....	6
2.2 Problemática común.....	6
2.3 Justificación.....	6
3. Objetivos.....	7
3.1 Objetivo general.....	7
3.2 Objetivos específicos.....	7
4. Marco teórico.....	8
4.1 Edge computing.....	8
4.2 Clústeres de cómputo.....	8
4.3 Alta disponibilidad y tolerancia a fallos.....	9
4.4 Modelos de lenguaje grandes y pequeños.....	9
4.5 Cuantización.....	9
4.6 Inferencia en CPU.....	10
4.7 Ollama y salidas estructuradas.....	10
4.8 Tareas de procesamiento de lenguaje natural.....	10
4.9 Componentes de infraestructura de software.....	11
4.10 Virtualización con adaptador puente.....	11
4.11 Ingeniería de instrucciones frente a entrenamiento.....	11
4.12 Monitoreo de sistemas.....	11
5. Proceso de diseño y evolución de la solución.....	12
5.1 Análisis de los casos de uso propuestos.....	12
5.2 Restricciones de hardware y decisión de usar una máquina virtual.....	12
5.3 Primer prototipo: chatbot de pedidos.....	12
5.4 Reevaluación del caso de uso.....	13
5.5 Prueba de extracción de pedidos.....	14
5.6 Prueba de clasificación de comentarios.....	14
5.7 Decisión final y principio de diseño.....	15
5.8 Iteraciones posteriores: análisis por tema, modo simple y confirmación.....	15
6. Arquitectura de la solución.....	17
6.1 Entorno del clúster: sistema operativo Linux.....	17
6.2 Integración del hardware.....	19
6.3 Topología y direccionamiento de red.....	20

6.4 Roles, servicios y puertos de cada nodo.....	23
7. Modelo de inteligencia artificial.....	24
7.1 Selección del modelo.....	24
7.2 Justificación de la elección.....	26
7.3 Distribución del modelo en el clúster.....	26
8. Implementación del sistema.....	27
8.1 Estructura del software.....	27
8.2 Ciclo de vida de una opinión.....	28
8.3 Interfaces de usuario.....	28
8.4 Generación y gestión de alertas.....	35
9. Tolerancia a fallos y alta disponibilidad.....	36
9.1 Detección de fallos.....	36
9.2 Escenarios de falla.....	36
9.3 Lo que ve cada usuario durante una falla.....	37
10. Demostración y experiencia de usuario.....	38
10.1 Flujo de uso del cliente.....	38
10.2 Flujo de uso del personal.....	38
10.3 Flujo de uso del gerente.....	38
11. Alcance de la solución.....	39
11.1 Lo que se logró resolver.....	39
11.2 Lo que no se logró o quedó fuera del alcance.....	39
11.3 Ventajas obtenidas.....	40
12. Limitaciones del hardware y recomendaciones.....	41
12.1 Limitaciones identificadas.....	41
12.2 Recomendaciones.....	42
13. Conclusiones técnicas.....	43
Referencias.....	44

Resumen ejecutivo

Este proyecto implementa un clúster de edge computing de dos nodos que clasifica opiniones de clientes con modelos de lenguaje ejecutados al 100 % en la red local, sin internet ni servicios en la nube. El clúster combina una Raspberry Pi 3 Model B de 1 GB de RAM con una máquina virtual Ubuntu Server alojada en una computadora de escritorio, ambos ejecutando modelos de la familia Qwen2.5 cuantizados a 4 bits mediante Ollama, solo con CPU.

El escenario de aplicación es una sucursal ficticia de restaurante, Comedor La Ceiba, ubicada en una carretera del interior de Guatemala con conectividad intermitente. Los clientes hacen su pedido y califican su visita en un kiosco de autoservicio con estrellas y un comentario libre, el personal de caja y los meseros también registran las quejas que reciben de forma verbal. Cada comentario se analiza automáticamente, el sistema identifica los temas que menciona (comida, servicio, tiempo de espera, limpieza, precio u otro), la opinión sobre cada tema y el sentimiento general. Un tablero en el monitor del gerente muestra los resultados en tiempo real, genera alertas cuando un problema se repite y permite marcarlas como "en proceso" o "resueltas". Dos páginas adicionales muestran el historial completo con filtros y el estado de cada nodo con métricas de memoria, procesador, temperatura, red e inferencia.

El trabajo se distribuye mediante una cola con prioridades. La máquina virtual, con el modelo de 3 mil millones de parámetros, es el worker principal y hace un análisis completo, tema por tema. La Raspberry Pi, con el modelo de 0.5 mil millones, colabora cuando la cola se acumula y asume todo el procesamiento si la máquina virtual falla, con un análisis simplificado cuyos resultados quedan marcados como provisionales. Cuando la máquina virtual vuelve a estar disponible, revisa automáticamente esas opiniones y las confirma o corrige. Así, la caída de un nodo no detiene el servicio ni provoca pérdida de opiniones, y ninguna decisión del gerente se basa en un análisis sin verificar, las quejas provisionales solo generan "posibles alertas", claramente diferenciadas.

La solución es el resultado de un proceso iterativo con pruebas reales en el hardware. El modelo de 0.5B falló como chatbot y como extractor de pedidos (acertó 1 de 4), pero clasificó comentarios de forma útil con un esquema de respuesta estricto. Pruebas posteriores mostraron que el mismo modelo no era confiable para asignar una opinión a cada tema, lo que llevó a separar un modo completo para la máquina virtual y un modo simple para la Raspberry Pi. El principio que guía todo el diseño es que el modelo interpreta el lenguaje y el código se encarga de contar, calcular y decidir.

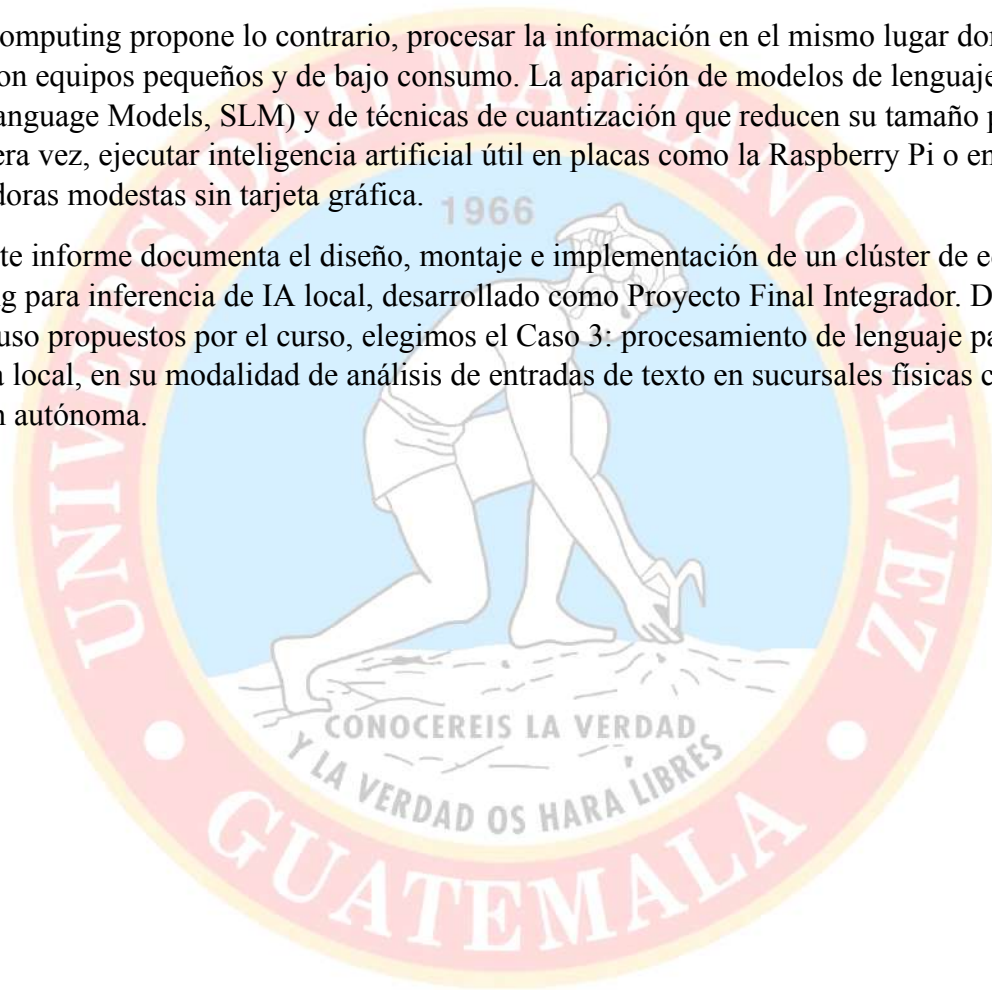
Entre las limitaciones identificadas están la memoria de 1 GB de la Raspberry Pi, el tiempo de carga en frío del modelo, la ausencia de reloj de hardware en la placa, la dependencia de la máquina virtual respecto al equipo anfitrión (incluido un conflicto con Hyper-V que ralentizaba la virtualización) y el hecho de que el nodo master es un punto único de falla. El informe documenta estas limitaciones y propone mejoras concretas.

1. Introducción

La inteligencia artificial generativa se ha popularizado principalmente a través de servicios en la nube, el usuario envía su consulta a un centro de datos remoto, un modelo de gran tamaño la procesa y la respuesta regresa por internet. Este modelo funciona bien en ciudades con buena conectividad, pero deja fuera a negocios ubicados en zonas donde el internet es lento, caro o intermitente. También implica costos recurrentes por consulta y el envío de datos de clientes a terceros.

El edge computing propone lo contrario, procesar la información en el mismo lugar donde se genera, con equipos pequeños y de bajo consumo. La aparición de modelos de lenguaje pequeños (Small Language Models, SLM) y de técnicas de cuantización que reducen su tamaño permite, por primera vez, ejecutar inteligencia artificial útil en placas como la Raspberry Pi o en computadoras modestas sin tarjeta gráfica.

El presente informe documenta el diseño, montaje e implementación de un clúster de edge computing para inferencia de IA local, desarrollado como Proyecto Final Integrador. De los tres casos de uso propuestos por el curso, elegimos el Caso 3: procesamiento de lenguaje para asistencia local, en su modalidad de análisis de entradas de texto en sucursales físicas con operación autónoma.



2. Definición del problema

2.1 Contexto

Muchos negocios de comida en Guatemala operan sucursales fuera de los centros urbanos, comedores en carreteras, restaurantes en cabeceras departamentales y puntos de venta en zonas turísticas del interior. En estos lugares la conectividad a internet suele depender de un solo proveedor, de enlaces inalámbricos o de datos móviles, y las interrupciones son frecuentes.

El proyecto trabaja con un escenario ficticio, Comedor La Ceiba, un restaurante con una sucursal sobre la carretera hacia Petén. La sucursal atiende a viajeros y transportistas, tiene horas pico marcadas y cuenta con un kiosco de autoservicio para que los clientes hagan sus pedidos sin hacer cola en caja.

2.2 Problemática común

La gerencia de la sucursal necesita saber qué opinan sus clientes para corregir problemas a tiempo: un baño descuidado, demoras en cocina, un platillo que llega frío o un precio que los clientes consideran alto. Hoy enfrenta tres dificultades:

1. **Las opiniones se pierden o no se analizan.** Los comentarios llegan de forma verbal a la cajera o al mesero, o se escriben en formularios que nadie revisa. Leer y ordenar cientos de comentarios a mano no es viable para un gerente que también atiende la operación.
2. **Los servicios de análisis en la nube no son una opción confiable.** Herramientas como ChatGPT o servicios de análisis de sentimiento requieren internet estable, cobran por consulta y envían los comentarios de los clientes a servidores de terceros.
3. **Los problemas se detectan tarde.** Sin un análisis continuo, una queja repetida, por ejemplo sobre la limpieza del baño, solo se descubre cuando ya afectó a muchos clientes.

2.3 Justificación

La solución combina tecnologías distribuidas e inteligencia artificial por razones concretas:

- **Inteligencia artificial**, porque los comentarios de los clientes están escritos en lenguaje natural, con errores de ortografía, expresiones locales y varias ideas en una misma frase ("la comida rica pero tardaron mucho"). Un formulario cerrado no capta esos matices; un modelo de lenguaje sí puede interpretarlos y convertirlos en categorías medibles.
- **Procesamiento local (edge)**, porque la sucursal debe operar de forma autónoma aunque no haya internet. Además, procesar en el sitio elimina el costo por consulta y mantiene los datos de los clientes dentro del negocio.
- **Clúster de varios nodos**, porque un solo equipo es un punto único de falla. Con dos nodos de procesamiento, la falla de uno no detiene el análisis, y en horas pico ambos pueden trabajar en paralelo.
- **Hardware de bajo consumo**, porque una sucursal pequeña no puede justificar servidores ni tarjetas gráficas. Una Raspberry Pi consume pocos vatios y puede permanecer encendida todo el día.

3. Objetivos

3.1 Objetivo general

Diseñar, montar e implementar un clúster de edge computing de bajo consumo que clasifique automáticamente las opiniones de los clientes de una sucursal mediante un modelo de lenguaje ejecutado de forma 100 % local, y que mantenga la continuidad del servicio ante la caída de un nodo de procesamiento.

3.2 Objetivos específicos

1. Montar un clúster físico de dos nodos Linux (una Raspberry Pi 3 Model B y una máquina virtual en modo puente) interconectados por red cableada, con direccionamiento IP estático en una subred privada de clase C.
2. Seleccionar y configurar modelos de lenguaje cuantizados a 4 bits que se ejecuten exclusivamente con CPU, de acuerdo con la memoria disponible en cada nodo.
3. Evaluar experimentalmente qué tareas de lenguaje (conversación, extracción de datos, clasificación y análisis por tema) puede resolver con confiabilidad cada modelo, y diseñar la solución con base en esa evidencia.
4. Implementar un mecanismo de distribución del trabajo mediante una cola con prioridades, que sume la capacidad de ambos nodos en momentos de alta carga.
5. Garantizar que la caída de un worker no provoque pérdida de opiniones ni errores visibles para el usuario, mediante detección de fallos, cancelación y reintento automático.
6. Asegurar la calidad de los resultados mediante la revisión automática, por el modelo principal, de las clasificaciones provisionales hechas por el nodo de respaldo.
7. Desarrollar las interfaces del sistema: un kiosco para clientes, una pantalla de registro para el personal, un tablero en tiempo real con alertas gestionables, un historial con filtros y una página de estado de los nodos.
8. Medir el desempeño del clúster (tiempos de respuesta, tokens por segundo, uso de memoria y procesador, temperatura, latencia, precisión y tiempo de recuperación ante fallos) y documentar las limitaciones del hardware utilizado.

4. Marco teórico

4.1 Edge computing

El edge computing es un modelo de computación en el que los datos se procesan cerca del lugar donde se generan, en lugar de enviarse a un centro de datos central o a la nube. En una arquitectura en la nube, un kiosco enviaría cada comentario a un servidor remoto; en una arquitectura edge, el comentario se procesa en un equipo ubicado dentro de la misma sucursal.

Las ventajas principales de este enfoque son cuatro:

- **Autonomía:** el sistema sigue funcionando aunque se pierda la conexión a internet.
- **Latencia:** la información no viaja por internet, así que el tiempo de respuesta depende solo de la red local y de la capacidad del equipo.
- **Privacidad:** los datos de los clientes no salen del negocio.
- **Costo:** no hay cobro por consulta ni dependencia de un proveedor externo.

La desventaja es que los equipos de borde tienen recursos limitados: poca memoria, procesadores modestos y normalmente sin tarjeta gráfica. Por eso, una solución edge exige elegir con cuidado qué tareas se ejecutan y con qué modelos.

4.2 Clústeres de cómputo

Un clúster es un conjunto de computadoras independientes, llamadas nodos, conectadas por una red y coordinadas para trabajar como un solo sistema. Según su propósito, los clústeres se clasifican en:

- **Clústeres de alto rendimiento:** dividen un cálculo grande en partes que se resuelven en paralelo.
- **Clústeres de balanceo de carga:** reparten muchas solicitudes independientes entre varios nodos para atender más trabajo por unidad de tiempo.
- **Clústeres de alta disponibilidad:** mantienen el servicio funcionando aunque falle uno de sus nodos.

En la mayoría de arquitecturas existe un **nodo master**, que recibe las solicitudes, guarda el estado y coordina, y uno o varios **nodos worker**, que ejecutan el trabajo pesado.

Conectar equipos en red no suma su memoria RAM, cada proceso sigue limitado a la memoria del nodo donde se ejecuta. Un modelo que necesita 2 GB de RAM no puede ejecutarse en una placa de 1 GB solo porque esté conectada a otro equipo. Lo que sí suma un clúster es la **capacidad de procesamiento total**: dos nodos pueden atender dos tareas al mismo tiempo, y si uno falla, el otro puede continuar. Existen técnicas experimentales para dividir un modelo de lenguaje entre varias máquinas, como el modo RPC de llama.cpp, pero la latencia de red y la poca memoria de una Raspberry Pi 3 las hacen poco prácticas para este proyecto.

4.3 Alta disponibilidad y tolerancia a fallos

La tolerancia a fallos es la capacidad de un sistema de seguir operando cuando uno de sus componentes deja de funcionar. Los conceptos clave son:

- **Punto único de falla (SPOF):** un componente cuya caída detiene todo el sistema.
- **Conmutación por error (failover):** el proceso por el que un componente de respaldo asume el trabajo de uno que falló.
- **Verificación de salud (health check):** consultas periódicas que el sistema hace a cada nodo para saber si sigue respondiendo.
- **Activo-pasivo:** un nodo trabaja y el otro solo espera para reemplazarlo.
- **Activo-activo:** todos los nodos trabajan a la vez y, si uno falla, los demás absorben su carga.

La solución de este proyecto usa un esquema activo-activo con prioridad: ambos nodos procesan, pero el de mayor calidad tiene preferencia y el otro interviene cuando hay carga acumulada o cuando el principal falla.

4.4 Modelos de lenguaje grandes y pequeños

Un modelo de lenguaje es una red neuronal, generalmente basada en la arquitectura Transformer, entrenada para predecir el siguiente fragmento de texto (token) a partir de los anteriores. Su tamaño se mide en parámetros, los valores numéricos que el modelo aprendió durante el entrenamiento. Los modelos comerciales más conocidos tienen cientos de miles de millones de parámetros y requieren centros de datos con tarjetas gráficas especializadas.

Los modelos de lenguaje pequeños (SLM) tienen entre unos cientos de millones y algunos miles de millones de parámetros. Son menos capaces de razonar y conversar, pero pueden ejecutarse en una computadora personal o en una placa de desarrollo. Este proyecto usa dos modelos de la familia Qwen2.5, desarrollada por el equipo Qwen de Alibaba: uno de 0.5 mil millones de parámetros (0.5B) y otro de 3 mil millones (3B).

4.5 Cuantización

La cuantización reduce la precisión numérica con la que se almacenan los parámetros de un modelo para disminuir su tamaño y su consumo de memoria. Un modelo en media precisión (FP16) usa 16 bits, es decir 2 bytes, por parámetro; cuantizado a 4 bits, usa alrededor de medio byte.

Un modelo de 3B en FP16 ocupa alrededor de 6 GB, mientras que a 4 bits baja a cerca de 1.5 GB más el espacio de algunas capas que se conservan con mayor precisión. En la práctica, el modelo qwen2.5:3b cuantizado ocupa cerca de 1.9 GB y el qwen2.5:0.5b alrededor de 400 MB. A esto se suma la memoria del contexto (la caché de atención), que crece con la longitud del texto procesado.

Los modelos usados se distribuyen en formato GGUF, propio del ecosistema llama.cpp, con un esquema de cuantización de 4 bits del tipo Q4_K_M. Este esquema agrupa los pesos en bloques y guarda factores de escala por bloque, lo que conserva buena parte de la calidad del modelo original con una fracción de su tamaño.

4.6 Inferencia en CPU

La inferencia es el uso de un modelo ya entrenado para producir una respuesta. Ocurre en dos fases:

1. **Procesamiento del prompt (prefill):** el modelo lee todo el texto de entrada. Cuanto más largo es el prompt (instrucciones, ejemplos y comentario), más tarda esta fase.
2. **Generación (decode):** el modelo produce la respuesta token por token. Su velocidad se mide en tokens por segundo y en CPU está limitada principalmente por el ancho de banda de la memoria.

Antes de la primera consulta, el modelo debe cargarse desde el almacenamiento a la RAM. Esta carga en frío puede tardar decenas de segundos en una tarjeta microSD. Por eso la solución mantiene los modelos cargados de forma permanente.

4.7 Ollama y salidas estructuradas

Ollama es una herramienta de código abierto que descarga, administra y ejecuta modelos de lenguaje de forma local. Internamente usa llama.cpp y expone una API REST en el puerto 11434. Entre sus parámetros relevantes para este proyecto están:

- **keep_alive:** cuánto tiempo permanece el modelo en memoria después de una consulta. Por defecto son 5 minutos; con el valor -1 se mantiene cargado indefinidamente.
- **temperature:** controla la aleatoriedad de la respuesta. Con 0, el modelo siempre elige la opción más probable, lo que produce resultados consistentes.
- **num_ctx:** el tamaño máximo del contexto en tokens, que determina cuánta memoria se reserva para la caché de atención.
- **format:** permite pasar un esquema JSON. Ollama convierte el esquema en una gramática que restringe los tokens que el modelo puede generar, de modo que la respuesta siempre cumple el formato y solo usa los valores permitidos. Esta técnica se conoce como decodificación restringida y es especialmente valiosa en modelos pequeños, que sin ella tienden a inventar campos o salirse del formato.

4.8 Tareas de procesamiento de lenguaje natural

No todas las tareas de lenguaje tienen la misma dificultad para un modelo pequeño. De menor a mayor exigencia:

1. **Clasificación:** asignar etiquetas de una lista cerrada, por ejemplo el tema y el sentimiento de un comentario. El modelo solo elige entre opciones.
2. **Extracción:** convertir texto libre en datos estructurados, como productos y cantidades de un pedido. El modelo debe identificar entidades, contar y razonar sobre el contexto.
3. **Generación con conocimiento:** responder preguntas o conversar a partir de información, como un chatbot de atención o un asistente que consulta un manual. Exige redactar, razonar y no inventar información.

4.9 Componentes de infraestructura de software

- **Cola de trabajo (patrón productor-consumidor):** los productores (kiosco, caja) depositan tareas en una cola y los consumidores (workers) las toman de una en una. Desacopla la recepción del procesamiento: el usuario no espera al modelo y ninguna tarea se pierde si un worker falla.
- **Proxy inverso:** un servidor que recibe las peticiones de los clientes y las reenvía a la aplicación interna. En este proyecto cumple esa función **nginx**.
- **Gestor de procesos:** un programa que mantiene en ejecución una aplicación, la reinicia si se cae y la arranca con el sistema. En este proyecto se usa **PM2**, mencionado en las especificaciones del curso como ejemplo de orquestador ligero.
- **SQLite:** un motor de base de datos relacional que guarda toda la información en un solo archivo, sin un servidor aparte. Es adecuado para equipos con pocos recursos.

4.10 Virtualización con adaptador puente

Una máquina virtual (VM) es un sistema operativo completo que se ejecuta sobre otro, llamado anfitrión o host. En modo de adaptador puente (bridged), la VM se conecta directamente a la red física como si fuera un equipo independiente: obtiene su propia dirección IP en la misma subred que los demás nodos y es visible para ellos. Esta configuración, recomendada por el catedrático, permite que la VM participe en el clúster como un nodo más.

4.11 Ingeniería de instrucciones frente a entrenamiento

Existen dos formas de adaptar un modelo de lenguaje a una tarea:

- **Entrenamiento o ajuste fino (*fine-tuning*):** se modifican los parámetros del modelo con miles de ejemplos etiquetados. Requiere datos, tiempo y normalmente una GPU.
- **Ingeniería de instrucciones (*prompt engineering*):** el modelo no cambia; se diseña cuidadosamente lo que se le pide, con instrucciones claras, ejemplos resueltos y restricciones de formato.

Este proyecto usa exclusivamente ingeniería de instrucciones. Los modelos no fueron entrenados ni modificados: se usan tal como los publica su fabricante. La adaptación a la tarea se logra con cuatro capas: instrucciones del sistema, ejemplos resueltos, un esquema JSON que restringe la respuesta y una validación en código (sección 7.4). Este enfoque es el adecuado para hardware de borde sin GPU.

4.12 Monitoreo de sistemas

El monitoreo consiste en recolectar de forma periódica métricas de los equipos (memoria, procesador, temperatura, latencia) y de la aplicación (tiempos de respuesta, errores) para conocer su estado y detectar problemas. Un agente es un programa pequeño que corre en cada equipo monitoreado y expone sus métricas por la red. Para no afectar el rendimiento del sistema que se observa, las lecturas deben ser ligeras y espaciadas: en este proyecto, una lectura del procesador cada 5 segundos y un registro consolidado por minuto.

5. Proceso de diseño y evolución de la solución

La solución final no fue nuestra primera idea, surgió de tres iteraciones en las que cada prueba real en el hardware descartó una alternativa. Esta sección documenta ese proceso porque explica por qué cada decisión del diseño final está respaldada por evidencia.

5.1 Análisis de los casos de uso propuestos

Caso	Requisito principal	Viabilidad con una Pi 3B de 1 GB
1. Visión por computadora	Detección de objetos en tiempo real con YOLO-nano o MobileNet y una cámara	Posible a pocos cuadros por segundo; la cámara conectada a un solo nodo complica la tolerancia a fallos
2. Mantenimiento predictivo	Clasificación de flujos de datos de sensores simulados	Muy viable: los modelos son ligeros y la distribución es sencilla
3. Procesamiento de lenguaje	Modelo de lenguaje cuantizado a 4 bits para voz o texto	Limitado: en 1 GB solo cabe un modelo de unos 0.5B parámetros

5.2 Restricciones de hardware y decisión de usar una máquina virtual

Contábamos con una sola Raspberry Pi y un presupuesto reducido. Se evaluaron opciones recicladas (una computadora de escritorio vieja, una laptop, un teléfono) y la compra de placas como la Orange Pi Zero 3, pero el plazo de entrega impedía importar hardware.

Las reglas nos permitieron virtualizar la capacidad de una computadora para formar el clúster, con la condición de que exista al menos un dispositivo físico en la red local que cumpla una función real dentro del sistema. Con esa condición, la arquitectura quedó así:

- **Raspberry Pi 3B:** nodo físico, master del clúster y worker de respaldo.
- **Máquina virtual Ubuntu Server con 4 GB de RAM:** alojada en una computadora de escritorio y configurada en modo puente, como worker principal con un modelo de mayor tamaño.

La VM ejecuta la inferencia únicamente con CPU. La tarjeta gráfica del equipo anfitrión no se usa, lo que respeta la restricción de hardware sin GPU dedicada.

5.3 Primer prototipo: chatbot de pedidos

La primera propuesta fue un chatbot en el que el cliente conversaba con un asistente para hacer su pedido. Se instaló Ollama en la Raspberry Pi con el modelo qwen2.5:0.5b, y se desarrolló una página web servida por nginx que mostraba la conversación y la velocidad de generación.

El prototipo funcionó a nivel técnico, la Pi generó respuestas a una velocidad de entre 4.6 y 5.1 tokens por segundo, suficiente para que el texto apareciera de forma fluida. Sin embargo, la calidad de las respuestas mostró problemas serios:

Pregunta del cliente	Respuesta del modelo	Problema
¿Qué venden?	Mencionó 3 de los 5 productos del menú	Información incompleta
Quiero un desayuno chapín y café, ¿cuánto sería?	Recomendó el pedido pero no dio el total	No respondió la pregunta
Eso quiero, ¿cuánto vale?	"Son de 35 quetzales"	Total incorrecto: lo correcto era Q45

Además, el modelo cometió errores gramaticales ("¿Qué deseas que comi?"). Se mejoraron las instrucciones con un formato numerado, un ejemplo resuelto y una temperatura de 0.3, lo que redujo los errores sin eliminarlos.

5.4 Reevaluación del caso de uso

Más allá de la calidad del modelo, identificamos un problema de diseño: un cliente frente a un kiosco prefiere tocar un menú que escribirle a un chatbot. Para elegir platillos basta una interfaz táctil, y la IA no aporta valor. Se planteó entonces una pregunta: ¿quién escribe texto libre de forma natural y en qué situación un formulario es más lento que escribir una frase?

Con esa pregunta evaluamos varias alternativas:

Alternativa	Quién escribe	Tarea del modelo	Dificultad para un modelo de 0.5B
Chatbot de pedidos para clientes	Cliente	Conversar y calcular	Alta
Asistente de capacitación para empleados nuevos	Empleado	Responder con base en un manual	Alta
Captura de pedidos recibidos por WhatsApp	Cajero	Extraer productos y cantidades	Media
Registro de mermas de cocina	Encargado	Extraer productos y motivos	Media
Reportes de fallas de equipo	Empleados	Clasificar área y urgencia	Baja

Alternativa	Quién escribe	Tarea del modelo	Dificultad para un modelo de 0.5B
Buzón de opiniones de clientes	Cliente o personal	Clasificar tema y sentimiento	Baja

5.5 Prueba de extracción de pedidos

Se pidió al modelo qwen2.5:0.5b, en modo de respuesta JSON, que extrajera el cliente, si el pedido era para llevar y la lista de productos de cuatro mensajes escritos como los enviaría un cliente real.

#	Mensaje	Productos	Cantidades	Otros campos
1	"me mandas 2 pollos uno sin papas y 3 refrescos pa llevar, a nombre de Ana"	Correctos	Incorrecta: 1 pollo en vez de 2	Inventó la clave "for_carreteras"
2	"un desayuno chapín para mí y para mi novia, con café ambos, quiero comer aquí"	Copió el menú completo	Incorrectas	"Para llevar" invertido
3	"una hamburguesa con papas para mi novia y pollo frito con papas para mí, pasará a recolectarlo"	Correctos	Correctas	Puso "true" como nombre del cliente
4	Pedido para cuatro amigos	Copió el menú completo	Incorrectas	Respuesta sin estructura válida en el primer intento

5.6 Prueba de clasificación de comentarios

Se repitió la evaluación con la tarea de clasificación, esta vez usando un esquema JSON estricto que solo permitía seis temas y tres sentimientos, y temperatura 0.

Comentario	Temas	Sentimiento	Tiempo
"la comida rica pero tardaron mucho"	comida, tiempo de espera (correcto)	negativo (lo ideal era mixto)	24.5 s
"el mesero fue muy amable y todo estaba limpio"	servicio, servicio (faltó limpieza; repetido)	positivo (correcto)	35.8 s

Comentario	Temas	Sentimiento	Tiempo
"muy caro para lo poco que sirven"	precio (correcto), servicio (de más)	negativo (correcto)	7.8 s
"excelente pollo, volveré"	comida (correcto), servicio (de más)	positivo (correcto)	8.2 s
"el baño estaba sucio y nadie atendía"	servicio (correcto); faltó limpieza	negativo (correcto)	8.6 s

El sentimiento fue útil en los 5 casos y el tema principal se identificó en 4 de 5. Los errores fueron menores y corregibles como temas agregados de más, temas repetidos y dificultad con la categoría "limpieza". Lo más importante es que, gracias al esquema estricto, el modelo no produjo una respuesta inválida. Las dos primeras pruebas tardaron entre 24 y 36 segundos por la carga del modelo en memoria; con el modelo ya cargado, cada clasificación tomó alrededor de 8 segundos.

5.7 Decisión final y principio de diseño

Con base en las pruebas, elegimos el buzón de opiniones con clasificación automática. El cliente califica su visita en el mismo kiosco donde hace su pedido, con estrellas y un comentario opcional, y el personal de caja puede registrar las quejas que recibe de forma verbal.

El modelo solo convierte un comentario en etiquetas. Guardar, contar, calcular promedios y decidir cuándo emitir una alerta lo hace el código de forma exacta e instantánea. Las debilidades observadas se corrigieron así:

- **Temas repetidos o inválidos:** el código elimina duplicados y descarta cualquier valor fuera de la lista.
- **Dificultad con algunas categorías:** las instrucciones incluyen una descripción de cada tema y dos ejemplos resueltos.
- **Precisión del modelo pequeño:** el modelo de 3B en la máquina virtual es el worker principal, el de 0.5B actúa como respaldo.
- **Carga en frío:** los modelos se precargan al iniciar y se mantienen en memoria de forma permanente.

5.8 Iteraciones posteriores: análisis por tema, modo simple y confirmación

Con el sistema funcionando en la Raspberry Pi, se observó que el modelo de 0.5B casi no reconocía los comentarios mixtos. En "la cajera me miró feo, pero la comida me gustó", por ejemplo, se quedaba con una sola de las dos partes. Para resolverlo se cambió la tarea a análisis por tema: el modelo indica la opinión sobre cada tema y el código calcula el sentimiento general.

Comentario	Sentimiento general	Opinión por tema	Tiempo
La comida estaba fría, pero estaba rica	Correcto (mixto)	Parcial: agregó "servicio"	9.0 s
La comida tenía un pelo, la bebida sí me gustó	Correcto (mixto)	Incorrecto: asignó la queja a "servicio"	9.6 s
El mesero me molestó, la comida estaba deliciosa	Incorrecto (positivo)	Incorrecto: servicio quedó positivo	9.3 s
La mesera me miró feo, pero la comida me gustó	Correcto (mixto)	Correcto	9.3 s
rico el café pero la mesa sucia	Incorrecto (negativo)	Incorrecto: faltó limpieza	65.9 s (carga en frío)

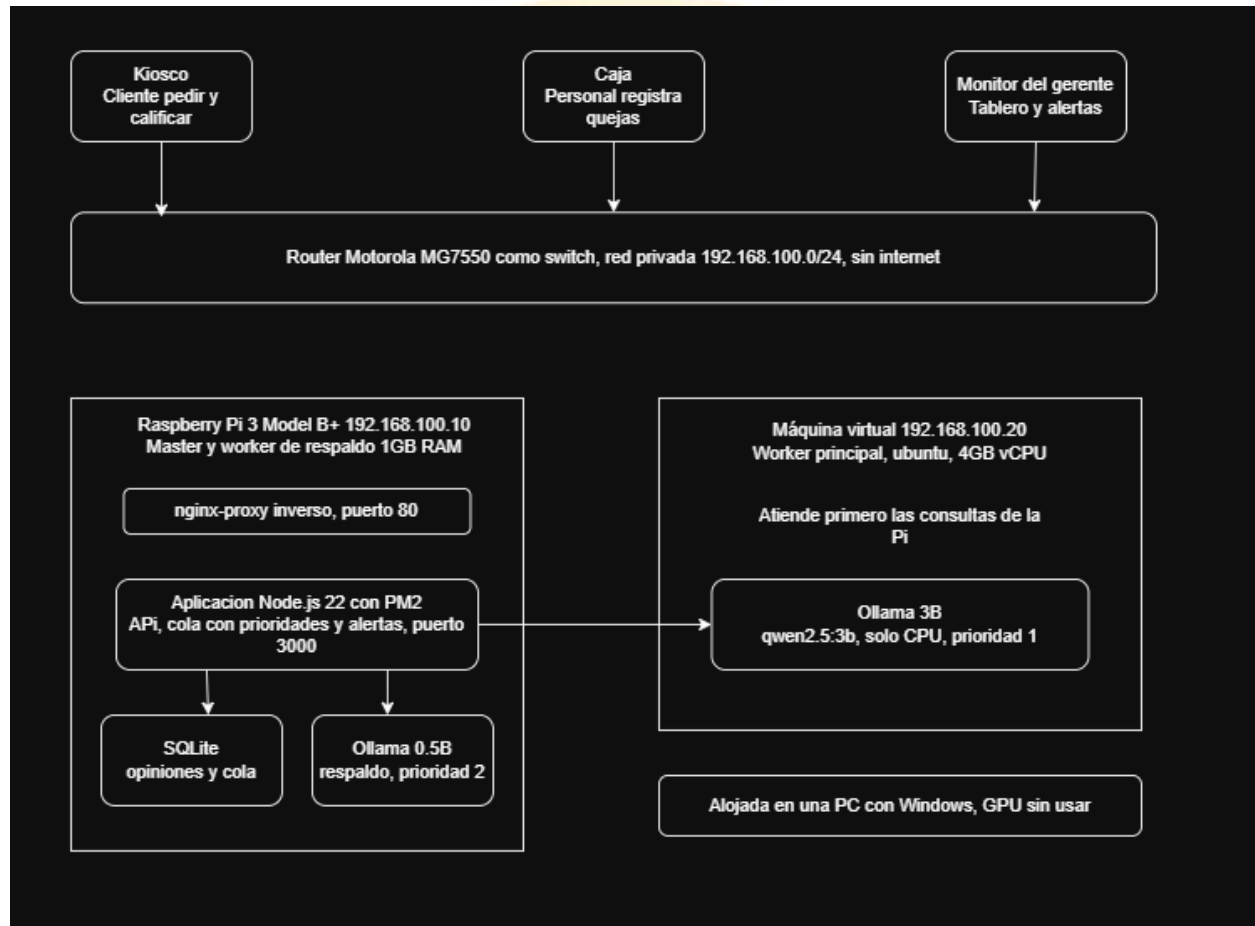
El sentimiento general mejoró en los comentarios mixtos (3 de 5 correctos, frente a casi ninguno antes), pero la opinión por tema fue correcta solo en 1 o 2 de 5, y el tiempo de clasificación se duplicó de unos 5 a unos 9 segundos, porque el modelo debía leer instrucciones más largas y escribir una respuesta más extensa.

Se evaluaron dos alternativas para el nodo de respaldo:

- **Repetir la clasificación varias veces y votar.** Se descartó porque multiplicaba el tiempo por 3 o 4 y solo corrige errores aleatorios, si el modelo asocia sistemáticamente "un pelo en la comida" con el servicio, repite el mismo error en cada votación.
- **Dividir el trabajo por calidad.** Fue la opción elegida, con tres decisiones:
 - a. La VM usa un modo completo (opinión por tema) y la Raspberry Pi un modo simple (un tema principal y un sentimiento general), que es más rápido y está dentro de su capacidad.
 - b. Todo lo que clasifica la Raspberry Pi queda como provisional, y la VM lo revisa automáticamente cuando está disponible, confirmándolo o corrigiendo.
 - c. Las quejas provisionales solo pueden generar posibles alertas, señaladas como "sin confirmar", para que las decisiones sobre el personal o la sucursal se base en un análisis sin verificar.

6. Arquitectura de la solución

El clúster está formado por dos nodos Linux conectados por cable a un router Motorola MG7550 usado como switch, sin conexión a internet: una Raspberry Pi 3 Model B que actúa como master y worker de respaldo, y una máquina virtual que actúa como worker principal. Los terminales de usuario (kiosco, caja y monitor del gerente) son navegadores web conectados a la misma red local.



Los terminales solo se comunican con la Raspberry Pi; la Pi decide qué worker clasifica cada opinión y consulta a la VM por la red local.

6.1 Entorno del clúster: sistema operativo Linux

Toda la infraestructura del clúster opera sobre Linux. La computadora anfitriona ejecuta Windows, pero su única función es alojar la máquina virtual, no ejecuta ningún componente del sistema.

Nodo	Sistema operativo	Motivo de la elección
Raspberry Pi 3 Model B	Raspberry Pi OS Lite de 64 bits (basado en Debian) 6.18.50	Sin escritorio, para liberar memoria; los 64 bits son obligatorios para ejecutar Ollama
Máquina virtual	Ubuntu Server 24.04.3 LTS	Sistema sin interfaz gráfica, ligero, con soporte de largo plazo y amplio soporte para Ollama

```

raspi@raspi:~$ cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 13 (trixie)"
NAME="Debian GNU/Linux"
VERSION_ID="13"
VERSION="13 (trixie)"
VERSION_CODENAME=trixie
DEBIAN_VERSION_FULL=13.7
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
raspi@raspi:~$ uname -a
Linux raspi 6.18.50+rpt-rpi-v8 #1 SMP PREEMPT Debian 1:6.18.50-1+rpt1 (2026-09-11) aarch64 GNU/Linux
raspi@raspi:~$

```

```

vm-worker@vm-worker:~$ cat /etc/os-release
PRETTY_NAME="Ubuntu 24.04.5 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.5 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
vm-worker@vm-worker:~$ hostnamectl
  Static hostname: vm-worker
            Icon name: computer-vm
            Chassis: vm
      Machine ID: d6419979311e43528c110da8f0994216
        Boot ID: cb2aabd069614703aceb3866a17175fa
  Virtualization: oracle
Operating System: Ubuntu 24.04.5 LTS
           Kernel: Linux 6.8.0-147-generic
   Architecture: x86_64
   Hardware Vendor: innotek GmbH
   Hardware Model: VirtualBox
Firmware Version: VirtualBox
   Firmware Date: Fri 2006-12-01
   Firmware Age: 19y 10month 2w 2d
vm-worker@vm-worker:~$

```

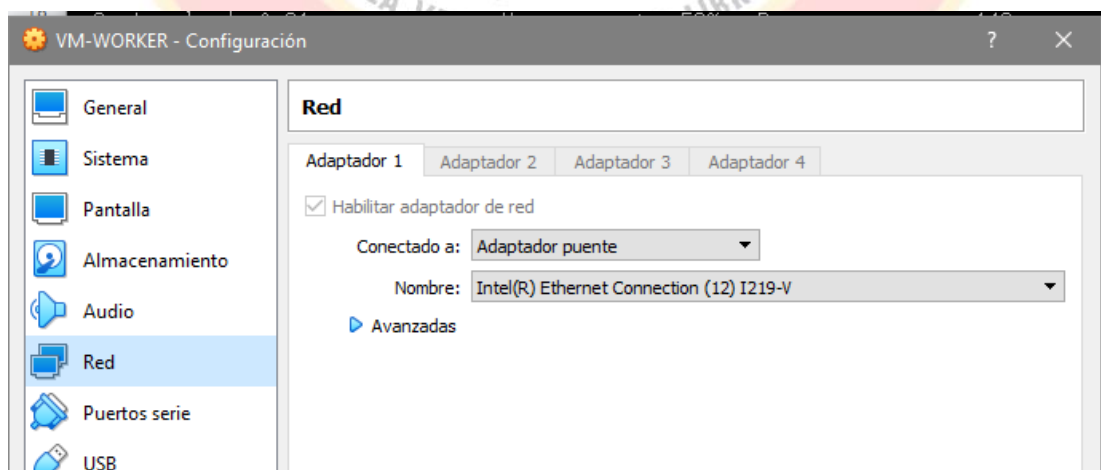
6.2 Integración del hardware

La arquitectura sigue el esquema indicado, una placa básica trabajando en conjunto con una computadora principal.

Componente	Raspberry Pi 3 Model B	Máquina virtual
Procesador	Broadcom BCM2837, 4 núcleos Cortex-A53 de 64 bits a 1.2 GHz	4 CPU virtuales con VT-x
Memoria RAM	1 GB LPDDR2	4 GB asignados
Almacenamiento	microSD de 32GB clase A1	Disco virtual dinámico de 40 GB
Red	Ethernet de 100 Mbps conectado por USB 2.0 (chip smsc95xx; unos 94 Mbps reales como máximo en TCP)	Adaptador puente hacia la tarjeta de red del anfitrión
GPU	-	-
Alimentación	Fuente de 5 V y 2.5 A, micro-USB	-

La Raspberry Pi utiliza disipadores para evitar que el procesador reduzca su frecuencia por temperatura durante la inferencia.

Configuración de la máquina virtual. En VirtualBox se creó la VM con 4 CPU virtuales, 4 GB de RAM, un disco dinámico de 40 GB y el adaptador de red en modo puente, para que obtenga su propia dirección IP en la red física como un equipo independiente.



6.3 Topología y direccionamiento de red

La topología final usa un router Motorola MG7550 como switch y una red privada de clase C dedicada, **192.168.100.0/24**, con direcciones IP estáticas y sin salida a internet. El cableado, en lugar de Wi-Fi, reduce la latencia y la variabilidad entre nodos.

Dispositivo	Dirección IP	Asignación	Función
Equipo anfitrión de la VM	192.168.100.5	Estática	Aloja la VM y sirve como terminal en la demostración
Raspberry Pi 3 Model B	192.168.100.10	Estática	Nodo master y worker de respaldo
Máquina virtual	192.168.100.20	Estática	Worker principal
Terminales adicionales	192.168.100.100 a 192.168.100.150	Automática (DHCP del router)	Kiosco, caja o monitor del gerente

Sincronización de hora. La Raspberry Pi 3 no tiene reloj de hardware, sin internet, al encenderse recupera la última hora guardada y no la real. La VM, en cambio, recibe la hora de Windows desde VirtualBox al arrancar. Para que los dos nodos compartan una hora correcta, la VM funciona como servidor de hora de la red del clúster utilizando la implementación chrony.

SSH Raspberry:

```
raspi@raspi:~$ timedatectl
    Local time: Fri 2026-10-16 21:46:32 CST
    Universal time: Sat 2026-10-17 03:46:32 UTC
        RTC time: n/a
        Time zone: America/Guatemala (CST, -0600)
System clock synchronized: yes
        NTP service: active
        RTC in local TZ: no
raspi@raspi:~$ timedatectl timesync-status
    Server: 192.168.100.20 (192.168.100.20)
Poll interval: 8min 32s (min: 32s; max 34min 8s)
    Leap: normal
    Version: 4
    Stratum: 10
    Reference: 7F7F0101
    Precision: 1us (-26)
Root distance: 0 (max: 5s)
    Offset: -44us
    Delay: 846us
    Jitter: 61us
    Packet count: 5
    Frequency: -0.565ppm
raspi@raspi:~$
```

SSH VM-Worker:

```
vm-worker@vm-worker:~$ chronyc clients
Hostname                NTP      Drop Int  IntL Last      Cmd      Drop Int  Last
=====
501 Not authorised
vm-worker@vm-worker:~$
```

Configuración de las IP estáticas. En la Raspberry Pi, que usa NetworkManager, se identifica la conexión cableada y se le asigna la dirección fija. En la VM, Ubuntu Server usa netplan. En Windows, la dirección se asigna en las propiedades IPv4 del adaptador Ethernet. En ningún caso se configura puerta de enlace en la red del clúster, porque no tiene salida a internet.

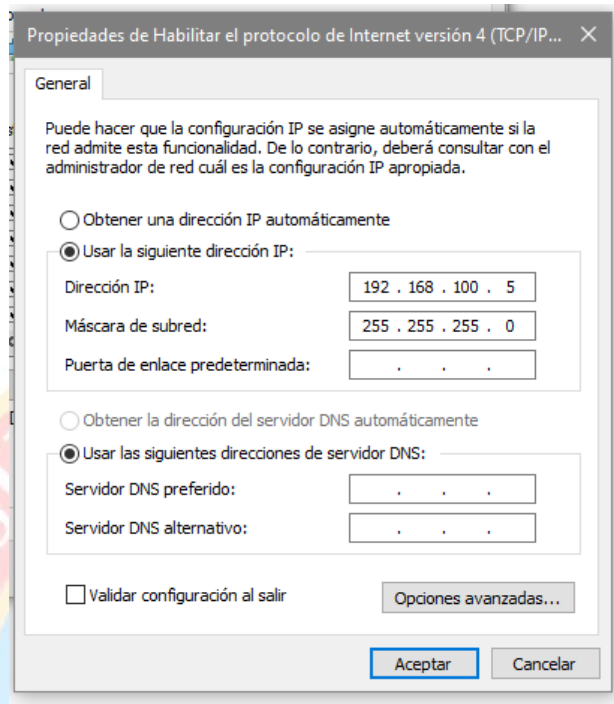
SSH Raspberry:

```
raspi@raspi:~$ ip a show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
    link/ether b8:27:eb:4c:42:07 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.10/24 brd 192.168.100.255 scope global noprefi
        valid_lft forever preferred_lft forever
    inet6 fe80::ba27:ebff:fe4c:4207/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
raspi@raspi:~$
```

SSH VM-Worker:

```
vm-worker@vm-worker:~$ ip a show enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
    link/ether 08:00:27:64:b0:21 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.20/24 brd 192.168.100.255 scope global enp0s3
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe64:b021/64 scope link
        valid_lft forever preferred_lft forever
vm-worker@vm-worker:~$
```

IP del ethernet físico:



Realizando prueba de comunicación de la Raspberry a la VM

```
raspi@raspi:~$ ping -c 5 192.168.100.20
PING 192.168.100.20 (192.168.100.20) 56(84) bytes of data.
64 bytes from 192.168.100.20: icmp_seq=1 ttl=64 time=1.03 ms
64 bytes from 192.168.100.20: icmp_seq=2 ttl=64 time=0.625 ms
64 bytes from 192.168.100.20: icmp_seq=3 ttl=64 time=0.648 ms
64 bytes from 192.168.100.20: icmp_seq=4 ttl=64 time=0.560 ms
64 bytes from 192.168.100.20: icmp_seq=5 ttl=64 time=0.601 ms

--- 192.168.100.20 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4074ms
rtt min/avg/max/mdev = 0.560/0.692/1.028/0.170 ms
raspi@raspi:~$
```

6.4 Roles, servicios y puertos de cada nodo

Nodo	Servicio	Puerto	Acceso	Función
Raspberry Pi	nginx	80	Red del clúster	Proxy inverso: recibe a los navegadores
Raspberry Pi	Aplicación Node.js 22 (administrada por PM2)	3000	Solo local	API, cola, workers, alertas y métricas
Raspberry Pi	Ollama con qwen2.5:0.5b	11434	Solo local (127.0.0.1)	Modelo de respaldo
Raspberry Pi	Base de datos SQLite	— (archivo)	Solo local en la Pi	Opiniones, pedidos, alertas, inferencias y métricas
Máquina virtual	Ollama con qwen2.5:3b	11434	Red del clúster	Modelo principal
Máquina virtual	Agente de métricas (Python)	9100	Red del clúster	Reporta RAM y CPU de la VM

Todos los servicios arrancan solos al encender el equipo: en la Pi, nginx y Ollama como servicios del sistema y la aplicación mediante PM2, en la VM, Ollama y el agente como servicios del sistema. La Raspberry Pi concentra la coordinación porque es el equipo físico, consume poca energía y puede permanecer encendida todo el día, como un equipo real de sucursal. La VM solo necesita Ollama y el agente, lo que permite reemplazarla por cualquier otro equipo con más capacidad.

Status del servicio PM2:

```

raspi@raspi:~ $ pm2 status

```

id	name	mode	status	cpu	memory
0	buzon	fork	online	0%	96.8mb

```

host metrics | cpu: 1% | ram usage: 78.4% | lo: 0.001mb/s 0.001mb/s | eth0: 0mb/s 0.001mb/s drop 8/min
raspi@raspi:~ $

```

7. Modelo de inteligencia artificial

La solución usa dos tamaños del mismo modelo, Qwen2.5 de 3B en la máquina virtual y Qwen2.5 de 0.5B en la Raspberry Pi, ambos cuantizados a 4 bits y ejecutados solo con CPU mediante Ollama. Cada uno trabaja en un modo distinto, ajustado a su capacidad.

7.1 Selección del modelo

Característica	qwen2.5:3b	qwen2.5:0.5b
Nodo	Máquina virtual	Raspberry Pi 3 Model B
Parámetros	3 mil millones	500 millones
Cuantización	4 bits, Q4_K_M	4 bits, Q4_K_M
Tamaño en disco	Alrededor de 1.9 GB	Alrededor de 400 MB
Memoria disponible en el nodo	4 GB asignados	1 GB en total
Modo de análisis	Completo: opinión por cada tema	Simple: tema principal y sentimiento general
Rol	Clasificación principal y revisión de provisionales	Respaldo y apoyo en picos de carga
Resultado	Confirmado	Provisional, hasta que la VM lo revise
Licencia	Qwen Research License	Apache 2.0
Contexto configurado	2048 tokens	2048 tokens

Los modelos se descargan con Ollama y quedan almacenados de forma persistente en cada nodo: en la tarjeta microSD de la Pi y en el disco virtual de la VM. No se requiere internet después de la descarga inicial.

Versión de ollama en Raspberry

```
raspi@raspi:~ $ ollama list
NAME                ID                SIZE      MODIFIED
asistente:latest    a8b0c5157701     397 MB    6 days ago
qwen2.5:0.5b        a8b0c5157701     397 MB    6 days ago
raspi@raspi:~ $ ollama show qwen2.5:0.5b
Model
  architecture      qwen2
  parameters         494.03M
  context length     32768
  embedding length    896
  quantization       Q4_K_M

Capabilities
  completion
  tools

System
  You are Qwen, created by Alibaba Cloud. You are a helpful assistant.

License
  Apache License
  Version 2.0, January 2004
  ...
```

Versión de ollama en VM

```
vm-worker@vm-worker:~$ ollama list
NAME                ID                SIZE      MODIFIED
qwen2.5:3b          357c53fb659c     1.9 GB    46 hours ago
vm-worker@vm-worker:~$ ollama show qwen2.5:3b
Model
  architecture      qwen2
  parameters         3.1B
  context length     32768
  embedding length    2048
  quantization       Q4_K_M

Capabilities
  completion
  tools

System
  You are Qwen, created by Alibaba Cloud. You are a helpful assistant.

License
  Qwen RESEARCH LICENSE AGREEMENT
  Qwen RESEARCH LICENSE AGREEMENT Release Date: September 19, 2024
  ...
vm-worker@vm-worker:~$
```

7.2 Justificación de la elección

1. **Ajuste de la memoria de cada nodo.** El modelo de 0.5B es el mayor que cabe con margen en la Raspberry Pi de 1 GB junto con el sistema operativo, nginx, Node.js y la base de datos; un modelo de 1B ocupa más de 1 GB cuantizado. En la VM, el modelo de 3B deja memoria libre para el sistema y el contexto.
2. **Mismo modelo en dos tamaños.** Usar la misma familia en ambos nodos mantiene un comportamiento coherente: los dos entienden el mismo tipo de instrucciones y los mismos temas.
3. **Soporte de español.** La familia Qwen2.5 fue entrenada con datos multilingües que incluyen el español, indispensable para interpretar comentarios con expresiones locales.
4. **Buen desempeño con salidas estructuradas.** Los modelos Qwen2.5 siguen bien los formatos JSON, y Ollama refuerza ese comportamiento con decodificación restringida por esquema.
5. **Modo de trabajo adecuado a cada tamaño.** Las pruebas (secciones 5.6 y 5.8) mostraron que el modelo de 3B puede hacer análisis por tema, mientras que el de 0.5B solo es confiable para identificar un tema principal y un sentimiento general.

Sobre la licencia: el modelo de 0.5B usa Apache 2.0, que permite uso comercial. El de 3B se publica bajo una licencia de investigación de Qwen, adecuada para este proyecto académico; una implementación comercial debería revisarla o sustituir el modelo.

7.3 Distribución del modelo en el clúster

Cada worker ejecuta una copia completa de su modelo, lo que se distribuye entre nodos son las tareas, no el modelo. El reparto sigue una cola con prioridades:

- **Worker principal (VM, prioridad 1):** toma cualquier opinión pendiente, en orden de llegada. Cuando no hay opiniones nuevas, revisa las provisionales.
- **Worker de respaldo (Pi, prioridad 2):** toma una opinión solo si la VM está fuera de línea o si la opinión lleva 8 segundos o más esperando en la cola. Sus resultados quedan como provisionales.

Situación	Comportamiento	Beneficio
Carga normal	La VM procesa todo, la Pi espera	Máxima precisión, con el modelo de 3B
Pico de carga	La VM procesa y la Pi toma las opiniones que se acumulan	Se suma la capacidad de los dos nodos
VM caída	La Pi procesa todo, en modo simple y como provisional	Continuidad del servicio
VM recuperada	La VM retoma la prioridad y revisa las provisionales	Se recupera la calidad

En un pico de carga, la capacidad combinada del clúster se aproxima a la suma de las capacidades de cada nodo.

8. Implementación del sistema

El software del clúster es una aplicación web en Node.js 22 sin dependencias externas: usa el módulo de SQLite integrado en Node.js, la función `fetch` nativa para comunicarse con Ollama y el servidor HTTP estándar. Esto reduce el consumo de memoria en la Raspberry Pi (el servidor completo ocupa alrededor de 80 MB) y simplifica la instalación. En la VM solo se agrega un agente de métricas escrito en Python con la biblioteca estándar.

8.1 Estructura del software

Archivo	Nodo	Función
server.js	Raspberry Pi	Servidor principal: API, cola, workers, revisión de provisionales, alertas, métricas e historial
esquema.js	Raspberry Pi	Estructura de la base de datos y migraciones automáticas
config.json	Raspberry Pi	Dirección, modelo, modo y prioridad de cada worker; umbrales de alertas; minutos para el aviso de caída
seed.js	Raspberry Pi	Carga datos de ejemplo de un día y una semana de operación (borra los existentes)
public/index.html	Raspberry Pi	Kiosco del cliente: hacer pedido o calificar la visita
public/caja.html	Raspberry Pi	Registro de opiniones por el personal
public/gerente.html	Raspberry Pi	Tablero del gerente
public/historial.html	Raspberry Pi	Historial completo de opiniones y pedidos
public/estado.html	Raspberry Pi	Estado y métricas de los nodos
default	Raspberry Pi	Configuración de nginx como proxy inverso
agente_vm.py	Máquina virtual	Agente que reporta RAM y CPU en el puerto 9100
agente-vm.service	Máquina virtual	Servicio del sistema para que el agente arranque solo

La configuración de los nodos se mantiene en un archivo separado para que un cambio de dirección IP o de modelo no requiera modificar el código: basta con editar `config.json` y ejecutar `pm2 restart buzón`.

8.2 Ciclo de vida de una opinión

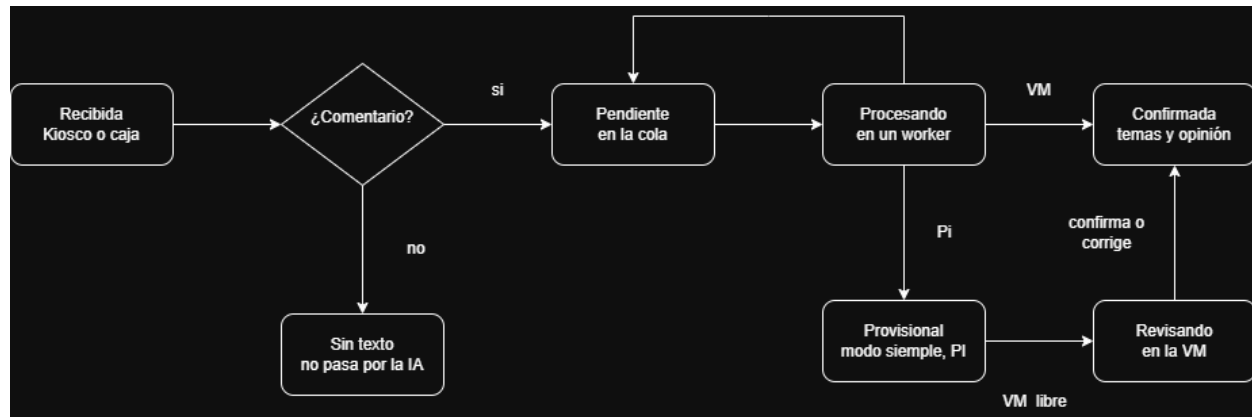


Figura 2. Estados de una opinión, incluida la revisión de los resultados provisionales

Si un worker falla durante la clasificación, la opinión vuelve a "pendiente" y otro worker la toma. Lo que clasifica la Raspberry Pi queda provisional hasta que la VM, cuando está libre, lo revisa y lo confirma o corrige.

Una opinión sin comentario, solo con estrellas, se guarda con estado "sin_texto" y no pasa por el modelo. Una opinión clasificada por la Pi queda provisional y, cuando la VM la toma para revisión, pasa temporalmente a "revisando" sin perder su resultado. Al reiniciar el servidor, toda opinión que quedó "procesando" regresa a "pendiente" y toda la que quedó "revisando" regresa a provisional, de modo que un corte de energía tampoco provoca pérdidas.

8.3 Interfaces de usuario

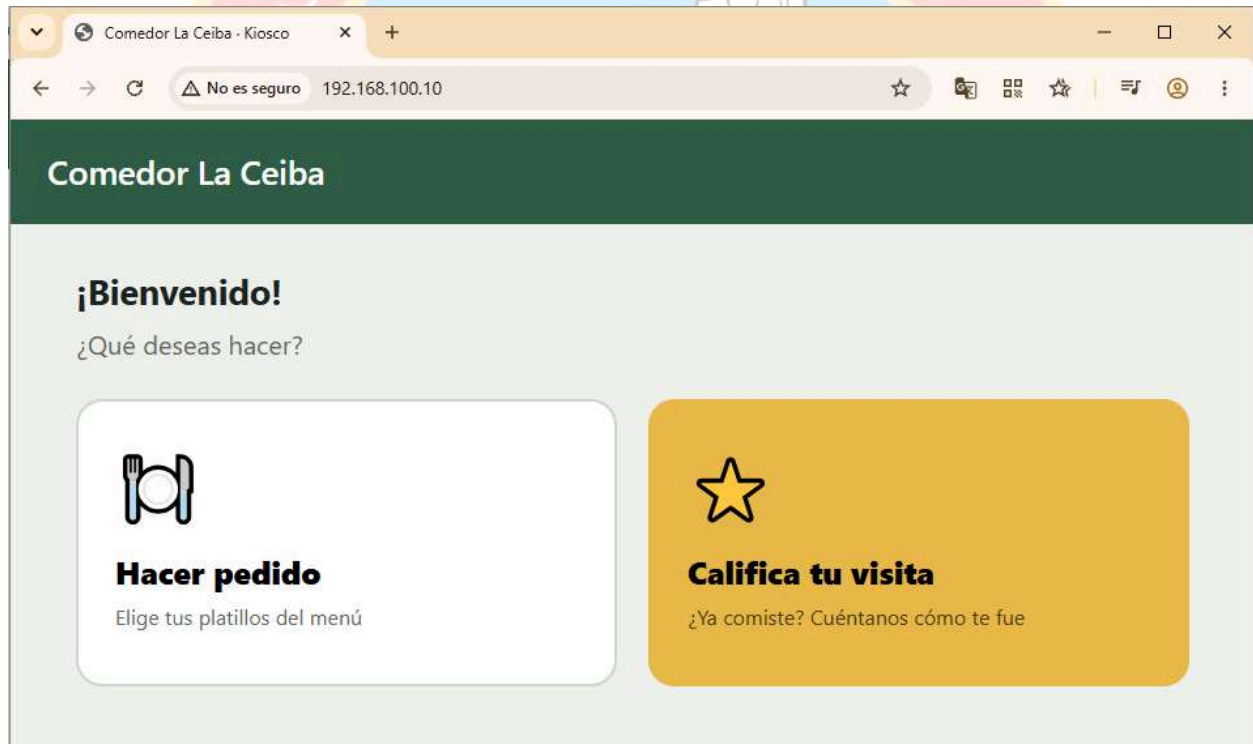
- Kiosco del cliente. La pantalla de inicio ofrece dos botones grandes, pensados para una pantalla táctil: "Hacer pedido" y "Califica tu visita". El pedido se arma tocando los productos, sin IA, y al confirmarlo se entrega un número de orden con un recordatorio para calificar la visita al terminar. La calificación tiene cinco estrellas y un comentario opcional; el cliente recibe un agradecimiento inmediato, sin esperar al modelo.
- Registro del personal. La cajera o el mesero indican quién registra, la calificación que dio el cliente (opcional) y el comentario con las palabras del cliente.
- Tablero del gerente. Se actualiza cada 3 segundos y muestra: el aviso del sistema cuando corresponde; las alertas, con sus botones de gestión y el historial de acciones; la calificación promedio, las opiniones del día, el porcentaje de positivas y las opiniones en cola o provisionales; el estado de cada nodo; los temas de los últimos 7 días con sus quejas; los pedidos del día con el total vendido; y las 15 opiniones más recientes. Cada tema aparece con + (positivo), - (negativo) o ± (mixto), y la columna "Procesado por" indica si el resultado es provisional, confirmado o corregido, mostrando en este último caso lo que había dicho la Pi.
- Historial. Muestra todas las opiniones y todos los pedidos en páginas de 50, con botones de anterior y siguiente, para que la Pi no lea miles de filas de una vez. Permite filtrar por

fechas, tema, sentimiento, nodo, origen y estado (confirmada, provisional, en cola, sin comentario o error), buscar texto sin importar tildes ni mayúsculas ("bano" encuentra "baño") y exportar a CSV con los mismos filtros.

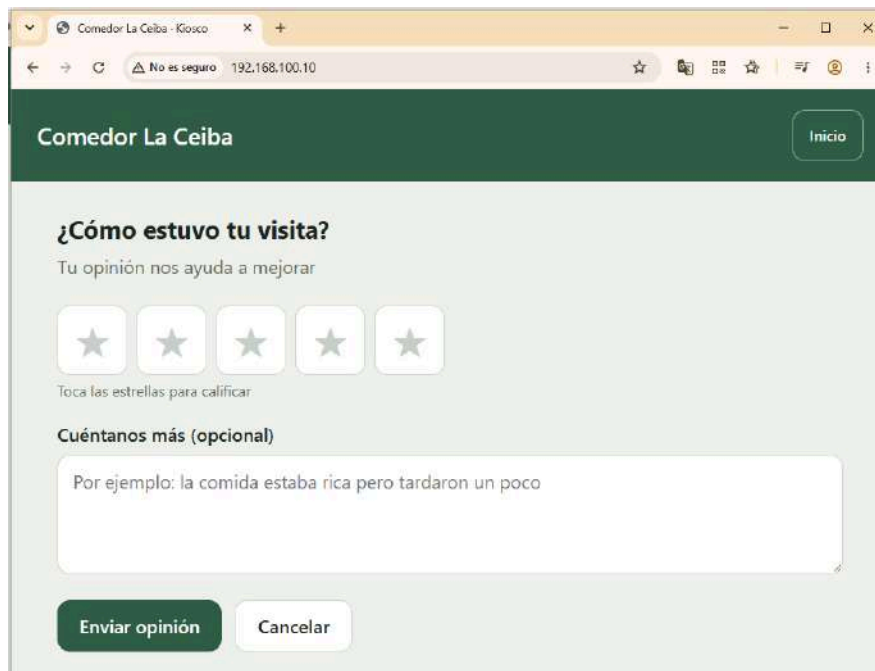
- Estado de los nodos. Muestra por nodo la RAM, el procesador y la temperatura (valor actual, promedio y máximo de la última hora), el modelo en memoria, la carga en frío, la latencia, las revisiones fallidas y las estadísticas de inferencia. Los datos más técnicos están plegados en "Detalles técnicos", y cada métrica tiene un botón "?" con su explicación.

Todas las interfaces funcionan sin internet: no cargan fuentes, librerías ni imágenes externas. El tablero y las demás páginas se abren desde cualquier navegador de la red local; el historial y el estado se abren en pestañas aparte desde los botones del tablero.

Kiosko Inicio:



Kiosko - Califica tu visita



Comedor La Ceiba Inicio

¿Cómo estuvo tu visita?
Tu opinión nos ayuda a mejorar

★ ★ ★ ★ ★

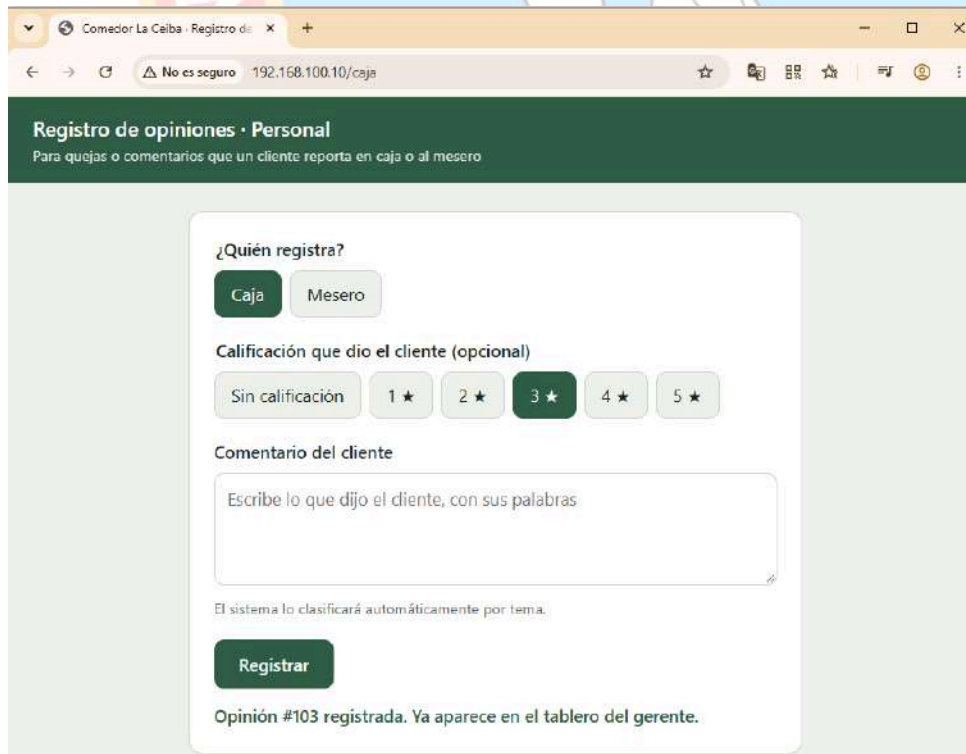
Toca las estrellas para calificar

Cuéntanos más (opcional)

Por ejemplo: la comida estaba rica pero tardaron un poco

Enviar opinión Cancelar

Registro de reseñas de parte del personal de trabajo:



Comedor La Ceiba - Registro de

Registro de opiniones - Personal
Para quejas o comentarios que un cliente reporta en caja o al mesero

¿Quién registra?

Caja Mesero

Calificación que dio el cliente (opcional)

Sin calificación 1 ★ 2 ★ 3 ★ 4 ★ 5 ★

Comentario del cliente

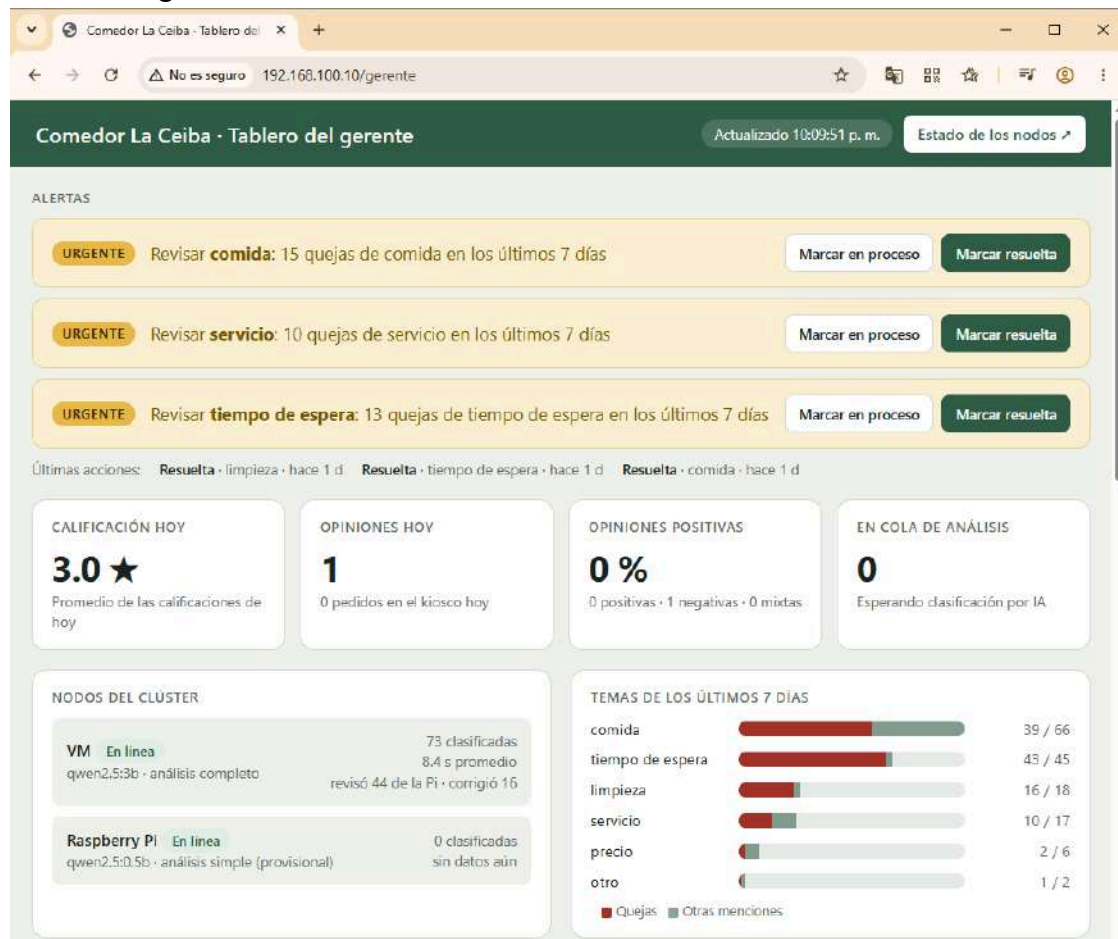
Escribe lo que dijo el cliente, con sus palabras

El sistema lo clasificará automáticamente por tema.

Registrar

Opinión #103 registrada. Ya aparece en el tablero del gerente.

Tablero del gerente:



la comida tardó — comida Antes (Rasp

GUATEMALA

Historial de reseñas y de órdenes:

Comedor La Ceiba · Historial

Ir al tablero

Opiniones

Pedidos

Desde

Hasta

Tema

Sentimiento

Estado

Nodo

dd/mm/aaaa

dd/mm/aaaa

Todos

Todos

Todos

Todos

Origen

Buscar en el comentario

Todos

Ej.: baño

Buscar

Limpiar

Exportar CSV

103 opiniones con estos filtros

Mostrando 1-50

#	FECHA	ORIGEN	ESTRELLAS	COMENTARIO	TEMAS	SENTIMIENTO	PROCESADO POR
103	16/10/26, 9:51 p. m.	caja	★ ★ ★	probando	- servicio	negativo	VM - 21.3 s
102	15/10/26, 9:45 p. m.	caja	★ ★ ★	la comida se tardó mucho	- tiempo de espera	negativo	CONFIRMADA VM - 4.2 s Antes (Raspberry Pi): tiempo de espera - negativo
101	15/10/26, 9:44 p. m.	caja	★ ★ ★	la comida estaba muy fría	- comida	negativo	CORREGIDA VM - 3.8 s Antes (Raspberry Pi): servicio - negativo

Comedor La Ceiba · Historial

Ir al tablero

Opiniones

Pedidos

Desde

Hasta

dd/mm/aaaa

dd/mm/aaaa

Buscar

Limpiar

Exportar CSV

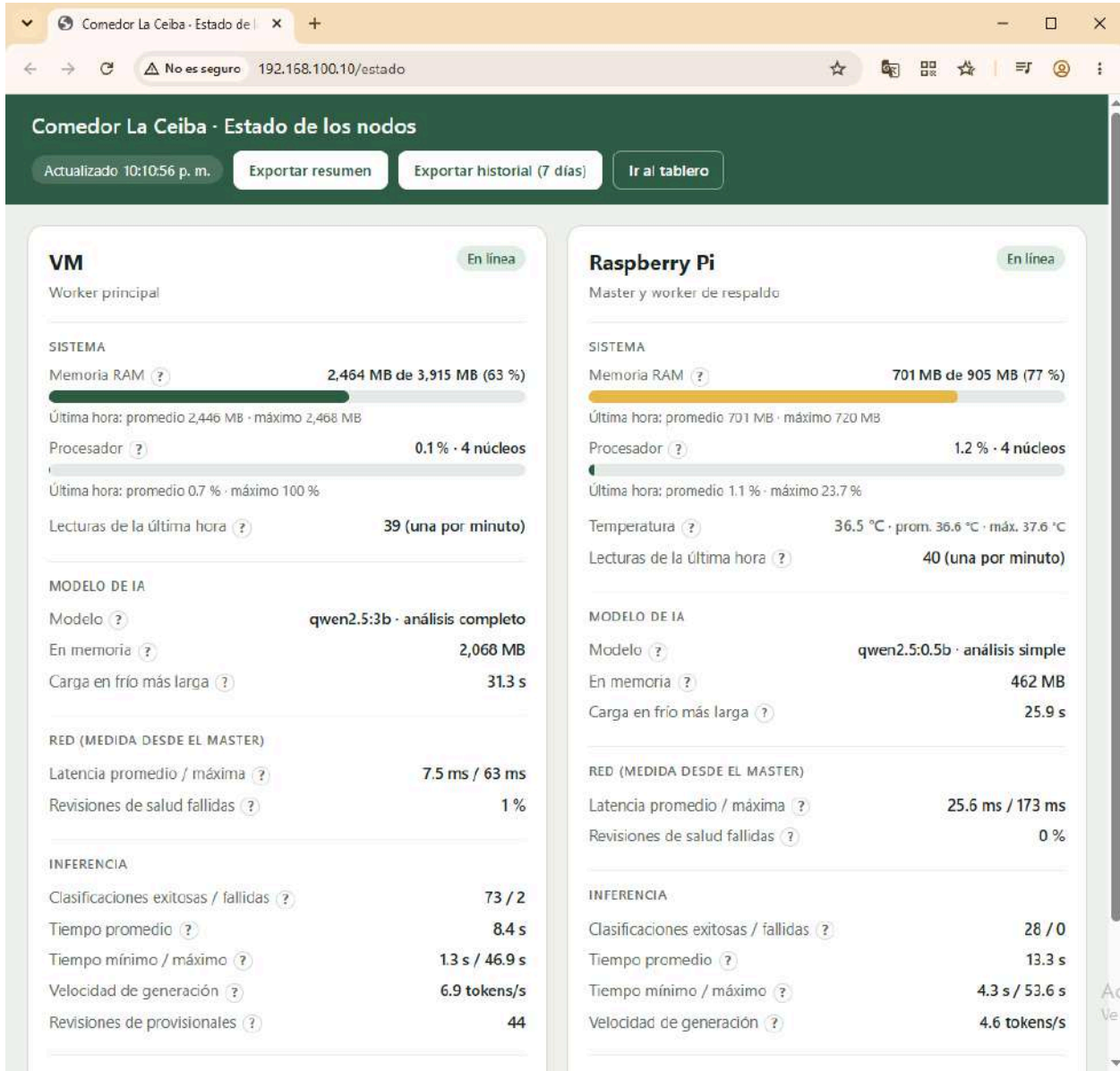
10 pedidos · Q694.00 vendidos con estos filtros

← Anterior

Página 1 de 1

Siguiente →

Parte técnica del estado de los nodos:



8.4 Generación y gestión de alertas

Las alertas se calculan con código mediante consultas a la base de datos, no con el modelo. Pedir al modelo que relea todas las opiniones sería lento y poco confiable; una consulta SQL es exacta e instantánea.

Una queja es un tema con opinión negativa dentro de una opinión clasificada. Con el análisis por tema, el comentario "la comida rica pero el baño sucio" cuenta como queja de limpieza, pero no de comida. Las reglas son:

Tipo	Condición	Aspecto en el tablero
Alerta	5 o más quejas confirmadas del mismo tema en el día	Franja amarilla con botones de gestión
Urgente	10 o más quejas confirmadas del mismo tema en 7 días	Franja roja con botones de gestión
Posible alerta	El umbral solo se alcanza sumando quejas provisionales	Franja morada punteada, "sin confirmar", sin botones

Las posibles alertas empiezan a contar de inmediato, sin esperar: retrasarlas no haría más preciso al modelo de respaldo y ocultaría un problema real. El riesgo de una decisión equivocada se controla porque requieren varias quejas, están marcadas como "sin confirmar" y piden leer los comentarios antes de actuar. Cuando la VM revisa las provisionales, la posible alerta se convierte en alerta normal o desaparece.

Gestión. Cada alerta confirmada tiene dos botones:

- **Marcar en proceso:** la alerta sigue visible, con la etiqueta "En proceso" y el tiempo transcurrido.
- **Marcar resuelta:** pide confirmación y la alerta desaparece. A partir de ese momento, las quejas de ese tema se vuelven a contar desde cero, si se acumulan de nuevo, la alerta reaparece, porque el problema volvió.

Cada acción queda en un historial visible debajo de las alertas.

Aviso del sistema. Una franja aparte, en color distinto al de las alertas de la sucursal, informa sobre el clúster mismo: si el nodo principal lleva 10 minutos o más fuera de línea, indica cuántos minutos y que los análisis nuevos son provisionales; cuando el nodo vuelve, indica cuántas opiniones provisionales está revisando. Los minutos se configuran en `config.json`.

Este mecanismo es lo que da a la sucursal operación autónoma: el sistema detecta por sí mismo un problema recurrente y lo señala, sin que nadie lea los comentarios uno por uno ni dependa de una oficina central.

9. Tolerancia a fallos y alta disponibilidad

Si el worker principal cae, incluso a mitad de una clasificación, ninguna opinión se pierde y el usuario no ve ningún error: la opinión regresa a la cola, el otro worker la procesa en segundos y, cuando el principal vuelve, revisa lo que hizo el respaldo.

9.1 Detección de fallos

Cada 3 segundos, el servidor consulta la ruta `/api/tags` de cada nodo con un tiempo límite de 3 segundos, y registra el tiempo de respuesta como medida de latencia. Un nodo se marca "fuera de línea" después de dos fallos consecutivos, lo que evita falsos positivos cuando un nodo responde lento por estar ocupado en una inferencia. Con estos valores, el tiempo de detección estimado es de 3 a 9 segundos, según el tipo de falla:

- Si el servicio se detiene, la conexión se rechaza de inmediato y la detección es más rápida.
- Si se desconecta el cable o se apaga la VM de golpe, cada verificación espera su tiempo límite y la detección tarda más.

Cuando el nodo vuelve a responder, se marca "en línea" en la siguiente verificación y se precarga su modelo.

9.2 Escenarios de falla

Escenario	Comportamiento del sistema	Efecto para el usuario
La VM cae sin trabajo en curso	La Pi asume las opiniones nuevas, como provisionales	Ninguno; el tablero las marca "Provisional"
La VM cae durante una clasificación	La consulta se cancela, la opinión vuelve a la cola y la Pi la procesa	Ninguno; esa opinión se clasifica unos segundos después
La VM cae con varias opiniones en cola	La Pi procesa la cola completa en orden de llegada	El tablero muestra la cola, que baja de forma gradual
La VM sigue caída 10 minutos o más	Aparece el aviso del sistema con los minutos de caída	El gerente sabe que los análisis son provisionales
Se acumulan quejas provisionales	Se genera una posible alerta, sin botones de acción	El gerente lee los comentarios antes de actuar
La VM se recupera	Retoma la prioridad y revisa las provisionales	Las filas cambian a "Confirmada" o "Corregida"
Falla el servicio de IA de la Pi	La VM continúa sola	Ninguno

Escenario	Comportamiento del sistema	Efecto para el usuario
Caen ambos workers	Las opiniones se siguen recibiendo y se acumulan en la cola	El kiosco funciona; la clasificación se reanuda al volver un worker
Se reinicia el servidor de la aplicación	PM2 lo levanta de nuevo; las opiniones en proceso vuelven a la cola	Interrupción de pocos segundos en las páginas
Cae la Raspberry Pi (master)	El sistema completo se detiene	Kiosco, caja y tablero dejan de responder

El último escenario es la principal limitación de la arquitectura: el nodo master es un punto único de falla. Las especificaciones exigen tolerar la caída de un **worker**, y ese requisito se cumple. La sección 12 propone cómo eliminar esta limitación.

9.3 Lo que ve cada usuario durante una falla

- **Cliente en el kiosco:** nada distinto. Recibe su "¡Gracias!" en cuanto envía la opinión.
- **Personal de caja:** nada distinto. Su registro se confirma con el número de opinión.
- **Gerente:** ve el cambio en el tablero. El nodo afectado aparece "Fuera de línea" con los minutos transcurridos, las opiniones nuevas aparecen como provisionales, las alertas que dependen de ellas aparecen como posibles y, si la caída se prolonga, el aviso del sistema lo informa. Cuando el nodo vuelve, ve cómo las provisionales se confirman o corrigen.

10. Demostración y experiencia de usuario

El sistema tiene tres tipos de usuario, cliente, personal y gerente, y cada uno interactúa con él desde un navegador en la red local, sin instalar nada. La Raspberry Pi y la VM no necesitan pantalla ni teclado: son servidores y se administran por SSH.

10.1 Flujo de uso del cliente

1. El cliente llega al kiosk y toca **"Hacer pedido"**.
2. Agrega productos con los botones + y -; el total se actualiza en pantalla.
3. Toca **"Confirmar pedido"** y recibe su número de orden, junto con la invitación a calificar su visita al terminar.
4. Después de comer, regresa al kiosk y toca **"Califica tu visita"**.
5. Elige de 1 a 5 estrellas; cada calificación muestra una leyenda ("Muy mala" a "Excelente").
6. Si lo desea, escribe un comentario, por ejemplo "la comida rica pero el baño estaba sucio".
7. Toca **"Enviar opinión"**, ve un mensaje de agradecimiento y el kiosk regresa solo al inicio.

10.2 Flujo de uso del personal

1. Un cliente le comenta a la cajera, o a un mesero, un problema durante su visita.
2. La cajera abre la pantalla de registro en su computadora de caja.
3. Indica quién registra (caja o mesero), la calificación que dio el cliente si la mencionó, y escribe el comentario con sus palabras.
4. Toca **"Registrar"** y recibe la confirmación con el número de opinión.

10.3 Flujo de uso del gerente

1. El gerente mantiene el tablero abierto en un monitor; se actualiza solo cada 3 segundos.
2. Arriba revisa el aviso del sistema, si lo hay, y las alertas: amarillas para problemas del día, rojas para problemas de la semana y moradas punteadas para posibles alertas sin confirmar.
3. Cuando asigna a alguien para atender un problema, pulsa **"Marcar en proceso"**; cuando el problema queda resuelto, pulsa **"Marcar resuelta"**.
4. Consulta los indicadores, los temas de la semana, los pedidos del día y las últimas opiniones, donde ve qué nodo procesó cada una y si el resultado es provisional, confirmado o corregido.
5. Para revisar opiniones anteriores, abre **"Ver todas"**: filtra, busca y, si lo necesita, exporta a CSV.
6. Para revisar el estado técnico del clúster, abre **"Estado de los nodos"**.

11. Alcance de la solución

11.1 Lo que se logró resolver

- Un clúster físico de dos nodos Linux, interconectados por cable en una subred privada con IP estáticas, que ejecuta inferencia de IA 100 % local y solo con CPU.
- El análisis automático de opiniones por tema y sentimiento, con dos modelos de lenguaje cuantizados a 4 bits, cada uno en el modo de análisis que su capacidad permite.
- La distribución del trabajo mediante una cola con prioridades, que suma la capacidad de ambos nodos en momentos de alta carga.
- La continuidad del servicio ante la caída del worker principal, sin pérdida de opiniones ni errores para el usuario, incluso durante una clasificación.
- La recuperación de la calidad después de una falla: el nodo principal revisa, confirma o corrige lo que procesó el respaldo.
- Un sistema completo con cinco interfaces (kiosco, personal, tablero, historial y estado de los nodos) que funciona sin internet.
- La detección automática de problemas recurrentes con alertas gestionables, y la protección contra decisiones basadas en resultados sin verificar mediante las posibles alertas.
- El monitoreo continuo del clúster, con métricas por minuto conservadas 7 días y exportables para su análisis.
- Una evaluación experimental de las capacidades reales de cada modelo, que respalda cada decisión de diseño.

11.2 Lo que no se logró o quedó fuera del alcance

Aspecto	Motivo
Chatbot conversacional para clientes	El modelo de 0.5B dio respuestas incompletas y cálculos incorrectos; además, un menú táctil resulta más práctico para el cliente
Extracción de pedidos desde texto libre	El modelo de 0.5B acertó 1 de 4 pedidos; el nodo de respaldo produciría datos inservibles
Análisis por tema en la Raspberry Pi	El modelo de 0.5B no empareja de forma confiable cada tema con su opinión; se usa un modo simple con resultados provisionales
Comandos de voz	El dictado de los navegadores envía el audio a la nube; procesarlo localmente con un modelo de voz competiría por los recursos de la Pi. El caso de uso permite elegir texto en lugar de voz
Alta disponibilidad del nodo master	Requiere replicar la base de datos y una dirección IP compartida entre nodos, lo que excedía el tiempo del proyecto

Aspecto	Motivo
Ejecutar un modelo dividido entre varios nodos	La memoria no se suma entre equipos; las técnicas experimentales son poco prácticas con la latencia de red y la memoria de una Pi 3
Entrenamiento o ajuste fino de los modelos	Requiere datos etiquetados y GPU; la ingeniería de instrucciones fue suficiente para la tarea
Autenticación de usuarios	No era necesaria para la demostración; es una mejora para un uso real
Integración con un sistema real de punto de venta	El módulo de pedidos es una simulación sencilla que da contexto al flujo de opiniones

11.3 Ventajas obtenidas

Aspecto	Análisis en la nube	Solución edge de este proyecto
Funcionamiento sin internet	No funciona	Funciona de forma completa
Costo por análisis	Cobro por consulta o suscripción	Ninguno después de la inversión inicial; incluso la revisión doble no tiene costo adicional
Privacidad de los datos	Los comentarios salen del negocio	Los comentarios no salen de la red local
Tolerancia a fallos	Depende del proveedor y del enlace	Propia: la falla de un worker no detiene el servicio
Calidad del modelo	Modelos grandes, de mayor precisión	Modelos pequeños, suficientes para clasificación, con revisión de los resultados del respaldo
Mantenimiento	A cargo del proveedor	A cargo del negocio, con monitoreo propio

La ventaja central es la autonomía: la sucursal analiza las opiniones de sus clientes y detecta problemas por sí misma, sin depender de internet, de terceros ni de una oficina central.

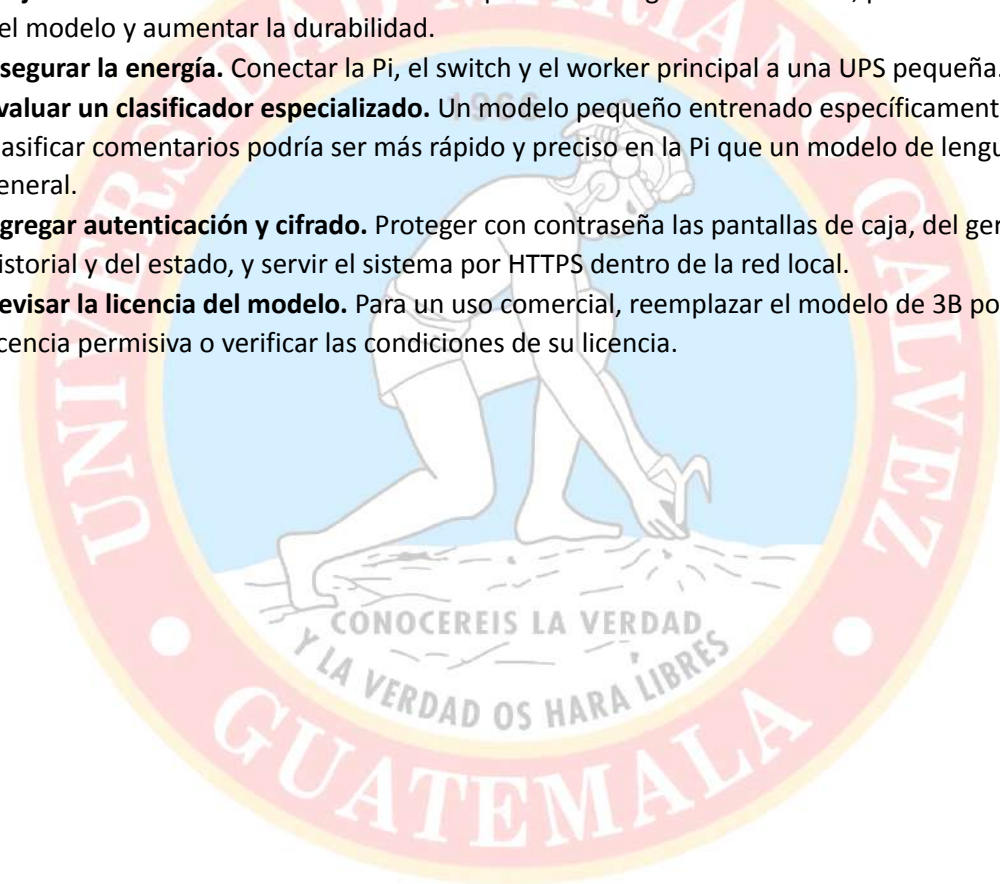
12. Limitaciones del hardware y recomendaciones

12.1 Limitaciones identificadas

Limitación	Efecto observado o esperado
1 GB de RAM en la Raspberry Pi 3 Model B	Solo admite un modelo de unos 0.5B parámetros, con menor precisión que el de 3B
Procesador Cortex-A53	Unos 4 a 6 segundos por clasificación en modo simple y unos 9 en modo completo; la CPU llega cerca del 100 % al clasificar
Lectura de la tarjeta microSD	La carga en frío del modelo tomó entre 24 y 66 segundos; se resolvió con la precarga y keep_alive -1
Ethernet de 100 Mbps conectado por USB 2.0	Tope práctico de unos 94 Mbps (enlace negociado a 100 Mbps/Full, confirmado con dmesg); no limita esta aplicación porque cada mensaje pesa pocos kilobytes
Sin reloj de hardware (RTC)	Sin internet, después de apagarse la Pi recupera la última hora guardada y no la real, lo que puede desfasar el conteo de quejas "del día"
Temperatura del procesador	Sin disipación, la Pi reduce su frecuencia cerca de 80 °C durante inferencias prolongadas
Nodo master único	Si la Pi falla, el sistema completo se detiene
VM dependiente del equipo anfitrión	El worker principal se detiene si la computadora se suspende, se reinicia por una actualización o se apaga
Conflicto de VirtualBox con Hyper-V	Con Hyper-V activo en Windows, la VM funcionó en un modo mucho más lento, con bloqueos del procesador virtual de más de un minuto
Direcciones por DHCP durante el desarrollo	La IP de la VM podía cambiar y había que actualizar la configuración; se elimina con las IP estáticas del montaje final
Precisión del modelo de respaldo	En las pruebas de análisis por tema acertó 1 o 2 de 5; mitigado con el modo simple, los resultados provisionales y la revisión de la VM

12.2 Recomendaciones

1. **Actualizar la placa.** Una Raspberry Pi 4 o 5 con 4 u 8 GB de RAM permitiría ejecutar el modelo de 3B en una placa física y prescindir de la máquina virtual. La Pi 5 incluye además un reloj de hardware con conector para batería.
2. **Resolver la hora sin internet.** Agregar un módulo RTC a la Pi 3 o configurar un servidor NTP local en otro equipo de la red.
3. **Eliminar el punto único de falla.** Agregar un segundo nodo master con la base de datos replicada y una dirección IP virtual compartida (por ejemplo, con keepalived).
4. **Usar un equipo dedicado como worker principal.** Una mini PC sin Windows evitaría los conflictos de virtualización y la dependencia de la computadora personal.
5. **Mejorar el almacenamiento.** Usar un SSD por USB en lugar de la microSD, para acelerar la carga del modelo y aumentar la durabilidad.
6. **Asegurar la energía.** Conectar la Pi, el switch y el worker principal a una UPS pequeña.
7. **Evaluar un clasificador especializado.** Un modelo pequeño entrenado específicamente para clasificar comentarios podría ser más rápido y preciso en la Pi que un modelo de lenguaje general.
8. **Agregar autenticación y cifrado.** Proteger con contraseña las pantallas de caja, del gerente, del historial y del estado, y servir el sistema por HTTPS dentro de la red local.
9. **Revisar la licencia del modelo.** Para un uso comercial, reemplazar el modelo de 3B por uno con licencia permisiva o verificar las condiciones de su licencia.



13. Conclusiones técnicas

1. **Es viable ejecutar inteligencia artificial útil en hardware de borde de bajo costo.** Una Raspberry Pi 3 Model B con 1 GB de RAM ejecuta un modelo de lenguaje de 0.5B parámetros cuantizado a 4 bits y produce clasificaciones útiles en pocos segundos, sin GPU ni internet.
2. El mismo modelo de 0.5B falló como chatbot, como extractor de pedidos y en el análisis por tema, pero resolvió bien la clasificación simple. Elegir una tarea adecuada a cada tamaño de modelo fue la decisión más importante del proyecto.
3. **Las salidas estructuradas son clave en modelos pequeños.** Restringir la respuesta con un esquema JSON eliminó por completo las respuestas inválidas en todas las pruebas realizadas, que eran el error más grave de las pruebas iniciales.
4. **No fue necesario entrenar los modelos.** Instrucciones claras, ejemplos resueltos, un esquema de respuesta y una validación en código bastaron para adaptarlos a la tarea, sin datos de entrenamiento ni GPU.
5. **Dividir responsabilidades entre el modelo y el código mejora la confiabilidad.** El modelo interpreta el lenguaje; el código valida, cuenta, calcula el sentimiento general y genera las alertas.
6. **La cola de trabajo es el mecanismo que permite la tolerancia a fallos.** Separar la recepción de la opinión de su clasificación permite responder al usuario de inmediato y garantiza que ninguna opinión se pierda.
7. **La disponibilidad y la calidad pueden separarse.** El nodo de respaldo garantiza que el servicio no se detenga; el nodo principal garantiza la calidad al revisar después lo que el respaldo hizo. Marcar los resultados como provisionales evita que una decisión humana dependa de un análisis sin verificar.
8. **El hardware y su entorno limitan la arquitectura y deben documentarse.** La memoria, el almacenamiento, la falta de reloj, el master único y hasta la configuración de virtualización del equipo anfitrión son restricciones reales que una implementación comercial tendría que resolver.

Referencias

- Alibaba Qwen Team. (2024). *Qwen2.5: A party of foundation models*. <https://qwenlm.github.io/blog/qwen2.5/>
- Qwen Team. (2024). *Qwen2.5 technical report* (arXiv:2412.15115). arXiv. <https://arxiv.org/abs/2412.15115>
- Ollama. (s. f.). *qwen2.5* [Biblioteca de modelos]. <https://ollama.com/library/qwen2.5>
- Ollama. (s. f.). *API documentation*. GitHub. <https://github.com/ollama/ollama/blob/main/docs/api.md>
- Ollama. (2024). *Structured outputs*. <https://ollama.com/blog/structured-outputs>
- ggml-org. (s. f.). *llama.cpp* [Repositorio de software]. GitHub. <https://github.com/ggml-org/llama.cpp>
- Raspberry Pi Ltd. (s. f.). *Raspberry Pi 3 Model B*. <https://www.raspberrypi.com/products/raspberry-pi-3-model-b/>
- Node.js. (s. f.). *SQLite* [Documentación de la API]. <https://nodejs.org/api/sqlite.html>
- SQLite. (s. f.). *JSON functions and operators*. <https://www.sqlite.org/json1.html>
- PM2. (s. f.). *Quick start*. <https://pm2.keymetrics.io/docs/usage/quick-start/>
- nginx. (s. f.). *Module ngx_http_proxy_module*. https://nginx.org/en/docs/http/ngx_http_proxy_module.html
- Shi, W., Cao, J., Zhang, Q., Li, Y. y Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5), 637-646.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. y Polosukhin, I. (2017). *Attention is all you need* (arXiv:1706.03762). arXiv. <https://arxiv.org/abs/1706.03762>