

Universidad Mariano Gálvez de Guatemala.

Sede Central, Plan sábado.

Ingeniería en Sistemas de Información y Ciencias de la Computación.

Curso: Arquitectura de sistemas II

Ciclo: 8

Sección: "A"



Nombre	Carnet	Correo
Justin Daniel Rivera López	0900-23-10643	jrival@miumg.edu.gt
Donalson José Calderón	0900-23-18946	dcalderonm6@miumg.edu.gt
Diego Gabriel Ventura Ochoa	0900-23-15244	dventura01@miumg.edu.gt
Jose Estuardo Aquino Quiquix	0900-13-1714	jaquinoq@miumg.edu.gt

Índice

Resumen ejecutivo.....	3
Introducción y planteamiento del problema	4
Identificación del Problema.....	4
Alcance	5
Objetivos:.....	5
Objetivos Específicos	5
Marco conceptual.....	5
IA en la Nube vs IA Local.....	5
Conceptos Claves	6
Arquitectura de la solución	7
Nodos y Red	8
Flujo de una petición de inferencia.....	8
Cómo funciona por dentro	10
1.1. La inferencia Local.....	11
El Servidor de Voz y la decisión de cada turno.....	12
La Memoria de CORE	13
Voz, Fecha, Cálculo e Internet.....	14
El Chat Web	14
El Cliente de voz de la Raspberry PI	15
El panel de control y los scripts	15
Seguridad y confidencialidad	16
Disponibilidad y tolerancia a fallos	16
Escenarios de prueba.....	16
Resultados y evaluación	17
Mediciones de latencia.....	18
Puertos.....	18
Orden de Arranque y Comprobación	19
Aplicaciones empresariales.....	19
Limitaciones y trabajo futuro.....	20
Conclusiones.....	21
Anexo A. Referencias.....	21

Resumen ejecutivo

Las organizaciones que incorporan asistentes de inteligencia artificial generativa a sus flujos de trabajo enfrentan una disyuntiva fundamental: la comodidad de los servicios en la nube viene acompañada de una pérdida de control sobre dónde se procesan y almacenan los datos internos. Cada consulta enviada a un proveedor externo implica que información potencialmente sensible contratos, datos de clientes, procedimientos internos— abandona el perímetro de la organización y queda sujeta a las políticas de retención y tratamiento de un tercero.

Este informe documenta el diseño, montaje, implementación y evaluación de CORE, un clúster de edge computing de dos nodos que demuestra que es posible ofrecer un asistente de IA funcional sin enviar datos a ningún proveedor externo. El sistema se construyó con hardware de borde de bajo costo (una mini PC con procesador Intel Core i5-10500T y una Raspberry Pi de 4 GB de RAM), sin GPU dedicada y sin conexión a internet para la etapa de inferencia.

La solución implementada se articula en cuatro capas:

Capa de inferencia: cada nodo ejecuta localmente el modelo llama3.2:1, un Small Language Model (SLM) de 1.2 mil millones de parámetros cuantizado a 4 bits (807 MB), mediante Ollama como motor de inferencia.

Capa de red: subred privada clase C (192.168.10.0/24) con direccionamiento IP estático, interconectada mediante un switch Gigabit Ethernet, sin puerta de enlace hacia internet.

Capa de orquestación: un middleware de balanceo de carga desarrollado a medida balanceador.py que implementa una estrategia de maestro con respaldo, toda petición se dirige primero al nodo más capaz (mini PC) y, solo si esta falla o no responde en 40 segundos, se redirige automáticamente a la Raspberry Pi.

Capa de interfaz: dos aplicaciones de usuario completas que consumen la capacidad de inferencia del clúster un chatbot web con registro de usuarios, verificación por correo e historial persistente, y un asistente de voz (CORE) con cara animada que integra transcripción con faster-whisper y síntesis de voz con Piper.

Introducción y planteamiento del problema

Contexto: Las herramientas de IA generativa en la nube se han convertido en un recurso cotidiano en el ámbito laboral. Sin embargo, su modelo de funcionamiento implica que cada consulta incluyendo el texto que el usuario escribe, los documentos que adjunta y las respuestas que recibe viaja a servidores que la organización no controla. Para áreas que gestionan información sensible (recursos humanos, legal, finanzas, atención al cliente), ese traslado representa un riesgo que muchas veces no está debidamente evaluado.

La IA local propone una alternativa: ejecutar el modelo dentro de los equipos de la propia organización. Este enfoque, habilitado por avances recientes en cuantización de modelos y por la disponibilidad de hardware de borde cada vez más capaz, permite mantener el control sobre los datos sin renunciar por completo a las capacidades de un asistente conversacional.

Identificación del Problema

Usar IA generativa en la nube con información interna expone a la organización a cuatro riesgos:

Riesgo	Descripción	Consecuencia
Confidencialidad	Cada pregunta, y lo que el usuario pegue en ella, sale de la red y queda en servidores de un tercero. La organización pierde el control sobre dónde se procesa y cuánto tiempo se conserva	Fuga potencial de información estratégica o sensible.
Cumplimiento	Las políticas internas y las normas de protección de datos pueden prohibir enviar cierta información a terceros	Sanciones legales, multas, daño reputacional.
Dependencia De red	Con una conexión deficiente o nula, como en una sucursal remota, el asistente deja de funcionar	Pérdida de productividad, imposibilidad de operar.
Punto único de falla	Si el único servidor local que ejecuta el modelo se cae, se pierde el servicio por completo.	Interrupción total del asistente.

Alcance

Este informe cubre el problema, la arquitectura, el funcionamiento interno, el código, la seguridad, la tolerancia a fallos, las limitaciones y las aplicaciones empresariales del clúster CORE. No compara el rendimiento con servicios comerciales específicos, ni evalúa modelos alternativos más allá del seleccionado.

Objetivos:

General: Demostrar que un asistente de IA puede operar dentro de la red de una organización, con hardware de borde sin GPU dedicada, protegiendo la confidencialidad de los datos y manteniendo la continuidad del servicio ante la falla de un equipo.

Objetivos Específicos

- Ejecutar de forma local un modelo de lenguaje pequeño cuantizado a 4 bits, sin enviar consultas a proveedores externos.
- Construir una red privada de dos nodos con direccionamiento IP estático, aislada de internet, interconectada mediante un Switch Gigabit Ethernet y un balanceador con respaldo automático.
- Ofrecer dos interfaces sobre el clúster: un chat web con usuarios y un asistente de voz.
- Implementar controles básicos de acceso: verificación de correo, contraseñas cifradas y aislamiento de conversaciones por usuario.
- Documentar el funcionamiento del código e identificar los límites de seguridad del montaje, con una mejora propuesta para cada uno.

Marco conceptual

IA en la Nube vs IA Local

La diferencia fundamental entre ambos enfoques radica en quién controla los datos y dónde se ejecuta el cómputo:

Criterio	IA en la nube	IA local (este proyecto)
Control de los datos	El proveedor procesa las consultas.	Las consultas no salen de la red.
Dependencia de Internet	Necesaria para consulta.	No es necesaria.
Capacidad del Modelo	Modelos muy grandes (cientos de miles de millones)	Modelos pequeños (1.2B parámetros).

Criterio	IA en la nube	IA local (este proyecto)
	de parámetros).	
Costo de Uso	Habitualmente por consulta o suscripción.	Hardware propio, sin cobro por consulta.
Mantenimiento	A cargo del proveedor.	A cargo de la organización.
Latencia	Depende de la red y del proveedor	Depende del hardware local.
Calidad de Respuesta	Alta, contextualmente rica.	Menor, limitada por el tamaño del modelo.

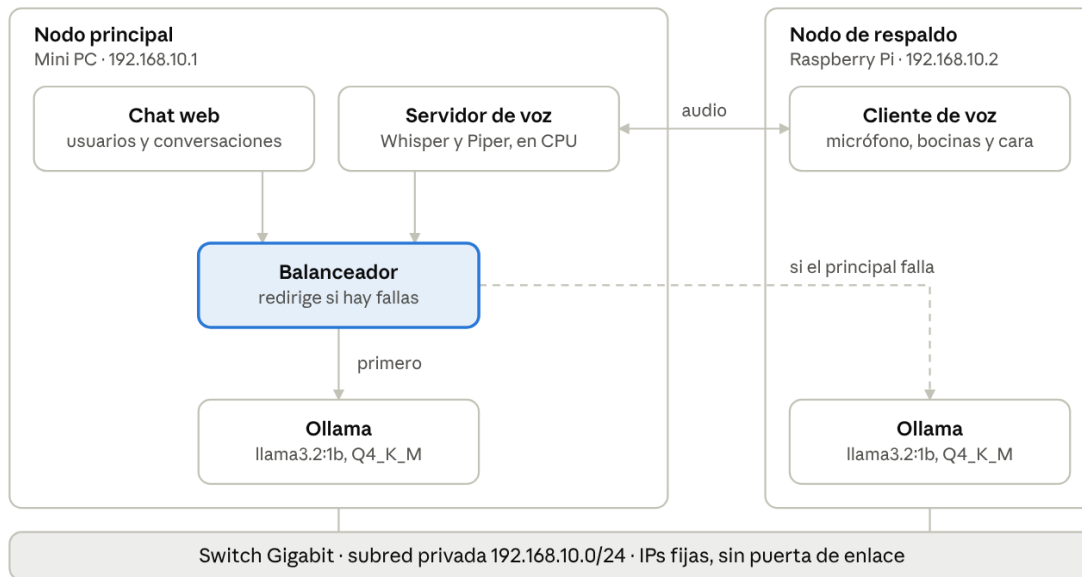
Conceptos Claves

- **Edge computing:** Paradigma de cómputo que procesa los datos cerca de donde se generan en el dispositivo o en hardware local cercano en lugar de enviarlos a servidores centralizados en la nube. Reduce la latencia, permite operación autónoma sin conectividad a internet, y mantiene los datos dentro del perímetro de la organización.
- **Small Language Model (SLM):** Versión reducida de un modelo de lenguaje grande (LLM), con menos parámetros (típicamente entre 1 y 7 mil millones), diseñada para ejecutarse en hardware de recursos limitados manteniendo una capacidad conversacional razonable. En este proyecto se utiliza llama3.2:1b, de 1.2 mil millones de parámetros.
- **Cuantización:** Técnica que reduce la precisión numérica de los parámetros (pesos) de un modelo de IA. Reduce su tamaño en memoria y acelera la inferencia, a cambio de una pérdida leve y aceptable de precisión. En este proyecto se utiliza la variante (cuantización a 4 bits), que redujo el tamaño del modelo de 1.3 GB (8 bits) a 807 MB.
- **Balanceo de carga:** Estrategia de distribuir el trabajo de procesamiento entre varios nodos para mejorar el rendimiento y la capacidad de respuesta.
- **Tolerancia a fallos:** Capacidad de un sistema para continuar ofreciendo el servicio ante la falla de uno o más de sus componentes. En este proyecto se implementa mediante la estrategia de maestro con respaldo.
- **Alta disponibilidad:** Diseño de sistemas orientado a garantizar que el servicio esté disponible la mayor parte del tiempo posible, minimizando las interrupciones.

Arquitectura de la solución

Visión General: CORE es un clúster de dos nodos que ejecuta el modelo de lenguaje, la transcripción de voz y la síntesis de voz dentro de una red privada. El código se compone de cinco programas independientes en Python (casi 5,000 líneas en total) que se comunican exclusivamente mediante HTTP y JSON.

Las consultas se procesan dentro de la red privada de dos nodos



El chat web y el servidor de voz envían todas las consultas al balanceador, que las manda a Ollama en la mini PC y, solo si esta falla, a Ollama en la Raspberry Pi. El cliente de voz de la Pi habla con el servidor de voz de la mini PC.

Nodos y Red

Nodo	Hardware	Dirección	Servicios
Principal	Mini PC, Intel Core i5-10500T (6 núcleos, 12 hilos, 2.30 GHz), 16 GB DDR4, gráficos Intel UHD 630 integrados	192.168.10.1	Balanceador (5000), chat web (8000), servidor de voz (8100), Ollama (11434), panel de control (5050, solo local)
Respaldo	Raspberry Pi, 4 GB de RAM	192.168.10.2	Ollama (11434), cliente de voz

Red: Ambos nodos se conectan físicamente mediante cable Ethernet a un switch Gigabit (1000 Mbps) de la marca Nexxt, formando una subred privada clase C dedicada y aislada. El direccionamiento IP es estático, configurado mediante NetworkManager, sin servidor DHCP en la red del clúster. No hay puerta de enlace hacia internet en la interfaz del clúster. Cada equipo tiene además su propia interfaz Wi-Fi, que se utiliza para instalar paquetes y para acceso remoto durante el desarrollo.

Homogeneidad del sistema operativo: Ambos nodos ejecutan Debian GNU/Linux 13 (trixie), con el mismo gestor de paquetes (apt), el mismo intérprete de Python (3.13) y el mismo gestor de servicios (systemd). La única diferencia es la arquitectura del procesador (x86-64 en el nodo maestro, ARM64 en el worker), inherente al hardware.

Flujo de una petición de inferencia

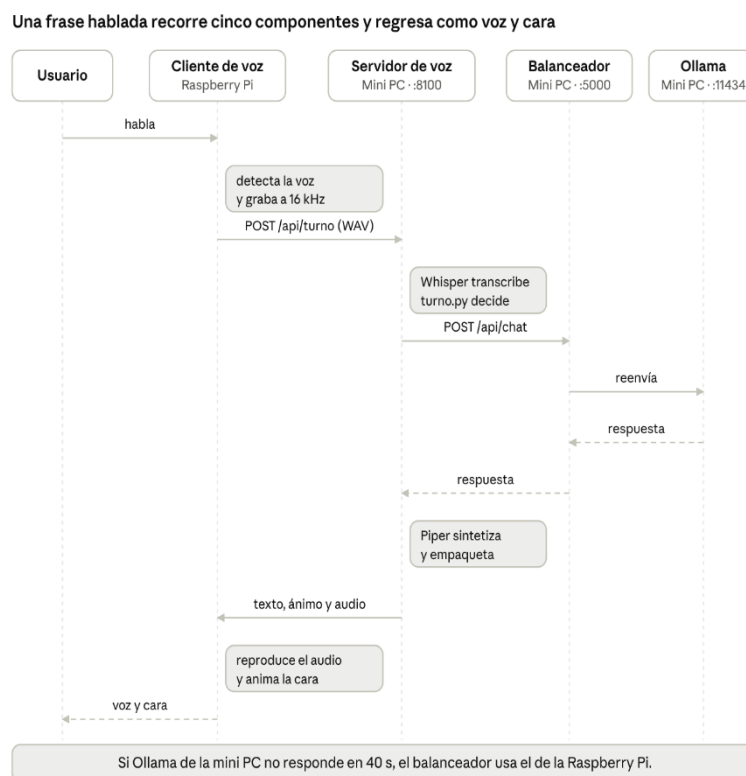
Flujo por texto (Chatbot web)

- El usuario escribe un mensaje en la interfaz web del chatbot (navegador).
- El backend del chatbot (FastAPI, puerto 8000) verifica la sesión del usuario y el límite de mensajes por minuto (15 mensajes/minuto).
- El backend envía la conversación completa (historial incluido) por HTTP al balanceador (puerto 5000).
- El balanceador reenvía la petición al nodo principal (Ollama local en 127.0.0.1:11434).
- Si el nodo principal no responde en 40 segundos o rechaza la conexión, el balanceador reintenta automáticamente con el nodo de respaldo (Raspberry Pi, 192.168.10.2:11434).

- La respuesta generada por el modelo regresa por la misma ruta hasta mostrarse en el navegador.
- El backend guarda la respuesta en la conversación del usuario (SQLite) y la muestra en el navegador.

Flujo por Voz (CORE)

- El micrófono conectado a la Raspberry Pi captura el audio del usuario (detección de voz por umbral de volumen cada 50 ms).
- El cliente de voz envía el audio al servidor de voz de CORE (mini PC, puerto 8100) mediante POST /api/turno.
- faster-whisper transcribe el audio a texto, ejecutado en CPU con cálculo en enteros de 8 bits.
- La lógica del asistente (turno.py) resuelve primero por reglas (calculadora, hora, pendientes, recordatorios); si ninguna aplica, consulta al modelo a través del balanceador.
- Piper convierte la respuesta en audio, ajustando velocidad y tono según el ánimo detectado.
- El servidor devuelve al cliente el texto, el estado de ánimo, la acción a ejecutar y el audio sintetizado en base64.
- La Raspberry Pi reproduce el audio y anima la cara según el estado de ánimo (inactivo, escuchando, procesando, hablando).



Cómo funciona por dentro

El Balanceador: el balanceador es un proxy HTTP en Python (Flask, cluster/balanceador.py) que se coloca entre las aplicaciones y Ollama. Recibe cada petición, elige un nodo y devuelve su respuesta como si viniera de un solo servidor.

Estrategia de distribución: maestro con respaldo. Inicialmente se evaluó un reparto por turnos (round-robin) parejo entre ambos nodos. Sin embargo, al existir una diferencia significativa de capacidad de cómputo entre la mini PC y la Raspberry Pi, se optó por una estrategia de Nodo Maestro con respaldo: toda petición nueva se intenta primero contra el nodo más capaz (mini PC); solo si este no responde dentro de un tiempo límite (40 segundos) o falla la conexión, el middleware redirige automáticamente la petición al nodo de respaldo (Raspberry Pi). Esta decisión de diseño constituye en sí misma una forma de redistribución de tareas documentada, conforme lo permite el enunciado del proyecto.

Funcionamiento del mecanismo de tolerancia a fallos. Cada petición entrante pasa por la función central `_consultar_cluster`, que recorre los nodos en orden de prioridad fijo:

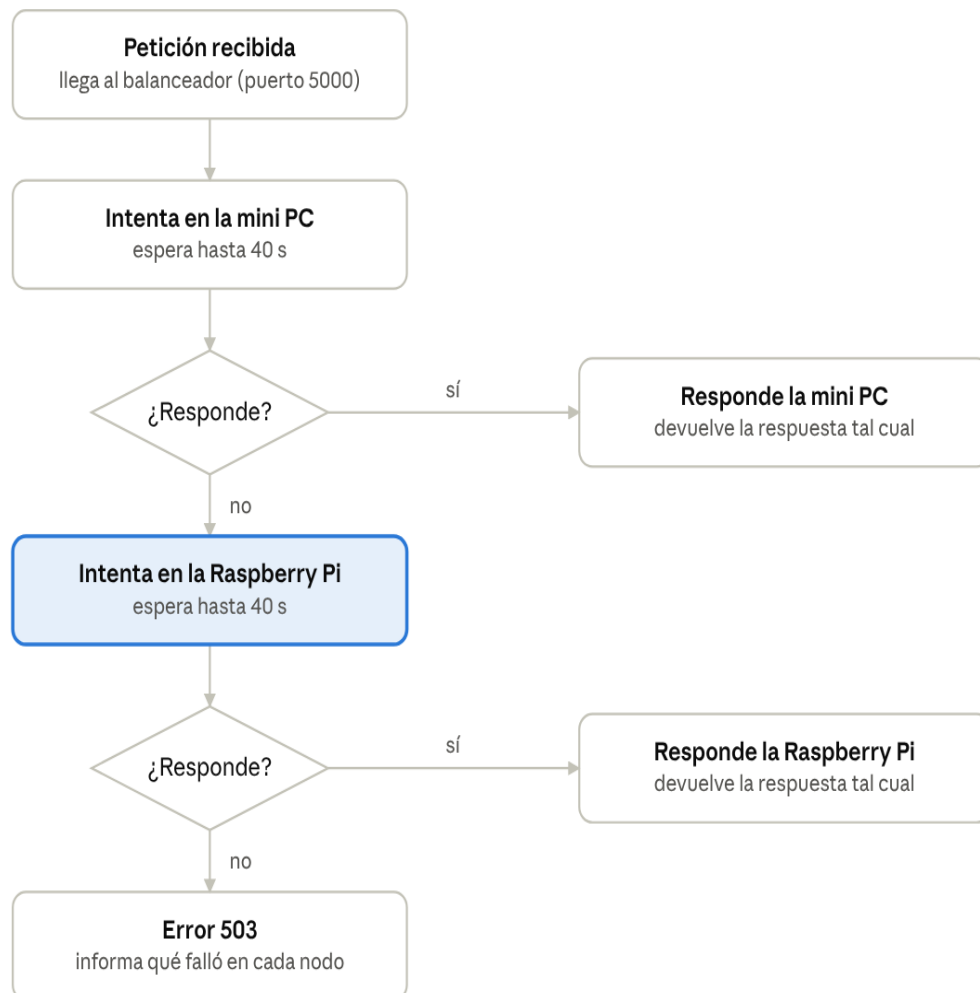
```
for nodo in _ORDEN_DE_PRIORIDAD:
    try:
        resp = requests.request(..., timeout=40)
        return resp # éxito, se entrega la respuesta
    except requests.exceptions.RequestException:
        continue # ese nodo falló, se intenta el siguiente
# Solo si TODOS los nodos fallan, se reporta un error 503
```

Si el nodo maestro no responde (tiempo agotado o conexión rechazada), el error se captura, se registra en el log, y la misma petición se reintenta automáticamente contra el siguiente nodo sin que el usuario final perciba una interrupción del servicio, más allá de un tiempo de espera adicional.

Consecuencias de este diseño:

- **No guarda memoria de los fallos:** Cada petición vuelve a probar primero la mini PC, así que esta recupera su trabajo en cuanto vuelve. Si queda colgada y no caída, cada petición espera 40 segundos antes de pasar a la Pi; una conexión rechazada, en cambio, se detecta al instante.
- **Solo reacciona a errores de conexión y a tiempos agotados:** Si Ollama contesta con un error HTTP (por ejemplo, 500), el balanceador lo reenvía sin probar el otro nodo.
- **No distingue un nodo caído de una respuesta lenta:** Con inferencia solo en CPU, una respuesta larga puede acercarse a los 40 segundos si los supera, la petición pasa a la Pi y la mini PC termina un trabajo que ya nadie espera.
- **Ambos nodos deben tener el mismo modelo,** porque el balanceador reenvía el nombre que pide la aplicación.

Cada petición prueba primero la mini PC y solo pasa a la Pi si no hay respuesta



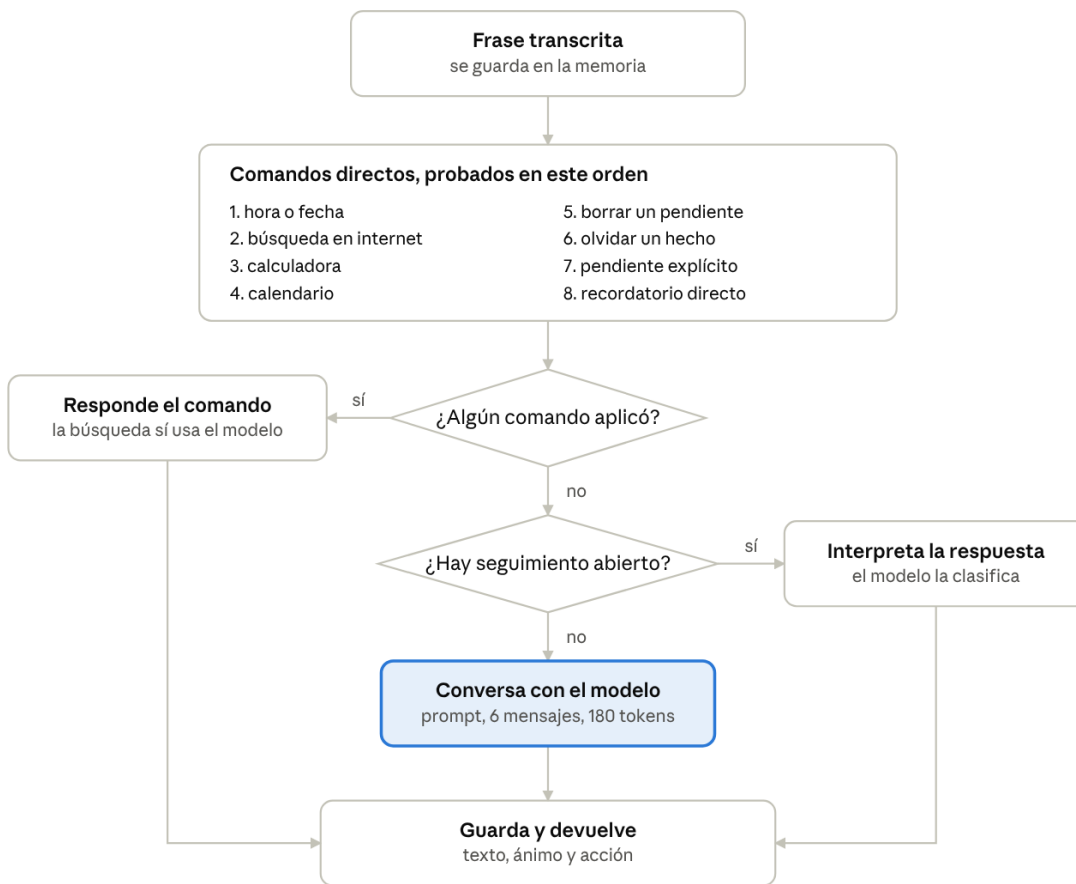
- **Carga del modelo:** La primera petición tras un periodo de inactividad es más lenta porque el modelo se copia a la memoria.
- **Generación:** El modelo produce la respuesta palabra por palabra (token por token), por lo que el tiempo crece con la longitud del texto.
- **Respuesta completa:** El chat, el servidor de voz y el balanceador piden la respuesta sin transmisión parcial (no streaming): el usuario la recibe cuando el modelo termina, y el límite de 40 segundos se cuenta hasta ese momento.
- **Contexto:** El servidor de voz pide un contexto de 6000 tokens y el chat no fija ninguno; es posible que Ollama recargue el modelo al alternar entre ambas aplicaciones, algo que no se midió.

El Servidor de Voz y la decisión de cada turno

main.py expone seis rutas: GET /api/salud, GET /api/saludo_inicial, POST /api/turno (la principal), GET /api/revisar_recordatorios, GET /api/pendientes y DELETE /api/pendientes/{id}. POST /api/turno recibe el audio, lo transcribe, llama a turno.procesar_turno y devuelve el texto, el ánimo, la acción y el audio sintetizado en base64. Imprime líneas [tiempos] con la latencia de cada etapa.

- Guarda el mensaje del usuario en la memoria.
- Prueba ocho comandos directos en orden fijo: hora o fecha, búsqueda en internet, calculadora, calendario, borrar pendiente, olvidar un hecho, pendiente explícito y recordatorio directo. Se reconocen por frases clave y el primero que aplica gana. Solo la búsqueda usa el modelo.
- Si ninguno aplicó y CORE acababa de preguntar si una tarea estaba hecha, interpreta la frase como esa respuesta.
- Si tampoco, conversa con el modelo: arma un prompt con hasta 15 hechos guardados y los pendientes, toma los últimos 6 mensajes, pide un máximo de 180 tokens y cada 5 mensajes añade una pregunta del banco. En segundo plano lanza una segunda consulta que extrae hechos nuevos del mensaje.
- Guarda la respuesta y la devuelve.

Un comando directo gana siempre; el modelo solo conversa si ninguno aplica



árbol de decisión de un turno · 2 decisiones, 3 salidas

El prompt de personalidad (`logica_asistente.py`) define el nombre, el idioma y un tono breve y sin markdown. Incluye reglas contra los errores típicos de un modelo pequeño: no inventar tareas ni citas, y no decir que es un modelo de IA ni que no tiene memoria. El ánimo que anima la cara se decide con palabras clave, no con IA.

La Memoria de CORE

`memoria.py` guarda todo en `asistente.db` con siete tablas: hechos, proyectos, conversacion, preguntas_hechas, pendientes, documentos y fragmentos. Las dos últimas, para búsqueda por significado, no se usan en el flujo de voz actual.

- **Hechos:** Antes de guardar uno nuevo se compara con los de su categoría; si se parece más de un 85%, se descarta, para no llenar la memoria de casi repetidos. CORE lee solo los 15 más recientes.

- **Historial:** obtener_historial_reciente devuelve los últimos mensajes en el formato que espera Ollama.
- **Recordatorios:** Un pendiente con fecha, activo y con notificado = 0 cuya fecha ya llegó se avisa una vez; si el usuario pide reprogramarlo, la fecha cambia y vuelve a quedar sin avisar.
- **Fechas:** Se guardan como hora local sin zona horaria, en texto ISO, así que dependen del reloj y la zona del equipo.

Voz, Fecha, Cálculo e Internet

- **Voz:** voz_servidor.py transcribe con el modelo small de Whisper, en CPU, con cálculo en enteros de 8 bits, decodificación directa y filtro de silencios. Sintetiza con Piper y cambia la velocidad y la variación del tono según el ánimo (por ejemplo, más rápida para "feliz" y más lenta para "triste"). Los dos modelos se cargan la primera vez que se usan, por lo que la primera respuesta es la más lenta.
- **Fechas:** tiempo.py entiende fechas con expresiones regulares, no con el modelo, en este orden: "en N minutos u horas", "mañana", "el 12 de octubre", "a las 17 horas con 53 minutos" y "a las 6:30 pm". Sin periodo, las horas de 1 a 7 se asumen de la tarde; si la hora ya pasó hoy, se pasa a mañana; "mañana" sin hora queda a las 9:00.
- **Cálculo:** calculadora.py cambia "más", "menos", "por" y "entre" por símbolos, aísla la expresión y la evalúa sin funciones disponibles, así que no puede ejecutar otro código. Responde con el formato $25 + 17 = 42$.
- **Internet:** internet.py comprueba la conexión, obtiene la hora real de worldtimeapi.org o, si no puede, usa el reloj del equipo y lo dice con honestidad, y busca en DuckDuckGo hasta 4 resultados que el modelo local redacta.

El Chat Web

- **Registro:** El usuario debe tener de 3 a 20 caracteres, la contraseña al menos 8, y ni el correo ni el usuario pueden repetirse. Se guarda un hash bcrypt, se crea un token de verificación de 24 horas y de un solo uso, y se envía por SMTP un enlace construido con URL_BASE.
- **Inicio de sesión:** El error es el mismo si el correo no existe o la contraseña es incorrecta. Tras 5 intentos fallidos la cuenta se bloquea 15 minutos. La cuenta debe estar verificada. Al entrar se crea una sesión de 7 días en una cookie HttpOnly y SameSite.

- **Chat:** Cada ruta comprueba que la conversación pertenezca al usuario. Un límite de 15 mensajes por minuto, guardado en la memoria del proceso, frena el abuso. El título de la conversación son los primeros 48 caracteres del primer mensaje. El navegador envía el historial completo en cada petición y el chat no usa prompt de sistema.
- **Datos:** Cinco tablas (usuarios, tokens_verificacion, sesiones, conversaciones y mensajes). Las consultas que reciben datos usan parámetros, no texto concatenado.
- **Interfaz:** Son cuatro archivos sin librerías externas. Un renderizador propio convierte el formato del modelo a HTML después de escapar el texto, así que una etiqueta <script> se muestra como texto. Una única regla @media adapta la barra lateral a celulares.

El Cliente de voz de la Raspberry PI

- **Detección de voz:** cliente_audio.py mide el volumen cada 50 ms. Si supera 0.02 empieza una frase; si pasa 1 segundo en silencio, la termina, y si duró menos de 0.4 segundos la descarta. Graba a la frecuencia nativa del micrófono y la convierte a 16 kHz; con las bocinas hace lo contrario.
- **Ciclo principal:** asistente_cliente.py espera al servidor, pide el saludo, y repite: comprobar recordatorios cada 20 segundos, escuchar, apagar el micrófono, enviar el audio (hasta 120 segundos), mostrar lo que entendió y reproducir la respuesta. Ejecuta la acción que indique el servidor: abrir o cerrar el calendario, refrescarlo o iniciar un cronómetro.
- **Cara y calendario:** cara.py dibuja ojos, cejas y boca unas 30 veces por segundo. El modo (inactivo, escuchando, procesando, hablando) cambia el color y el comportamiento; el ánimo cambia las cejas y la boca. calendario.py muestra los pendientes con fecha primero y un botón para borrarlos.

El panel de control y los scripts

panel_universidad.py es una aplicación Flask que escucha solo en 127.0.0.1:5050 y ofrece tres acciones por servicio: arrancar, detener y validar. Los servicios de la mini PC se lanzan como procesos en segundo plano con su registro en ~/IA/logs/ y se encuentran por el texto de su línea de comandos. Los de la Pi, el cliente de voz y el acceso público, se manejan por SSH con paramiko. Para administrarlo desde otra máquina se usa un túnel SSH. Los scripts abrir_panel.sh y conectar_pi.sh, y dos .bat de Windows que abren el túnel y las ventanas de registro, completan el arranque.

Seguridad y confidencialidad

El control más sólido del proyecto es arquitectónico: las consultas se procesan dentro de la red y no se envían a un proveedor de IA. Los controles internos son básicos y tienen límites que se declaran a continuación.

Tres detalles de diseño refuerzan esa base:

- Las contraseñas nunca se guardan, solo su hash (bcrypt).
- Los enlaces de verificación y las sesiones usan tokens aleatorios con caducidad.
- El texto que escribe el usuario o devuelve el modelo se muestra escapado, sin posibilidad de inyectar código en la página.

Limitación importante: En una red sin internet, el registro de usuarios nuevos no funciona porque depende del correo. Las cuentas deben crearse de antemano.

Disponibilidad y tolerancia a fallos

Estrategia: El balanceador envía cada petición a la mini PC y solo recurre a la Raspberry Pi si la mini PC falla o no responde en 40 segundos. La estrategia de maestro con respaldo redistribuye las tareas ante una falla sin penalizar el funcionamiento normal.

La tolerancia a fallos protege contra la caída de un nodo. El sistema también depende del switch, de la corriente y del propio balanceador, que corre en la mini PC: si esa máquina se apaga entera, el servicio se interrumpe. El respaldo cubre la inferencia, no el balanceador.

Escenarios de prueba

Escenario	Acción	Resultado esperado	Estado
A. Cae el nodo de respaldo	Detener Ollama en la Raspberry Pi	El chat sigue respondiendo y /salud marca raspberry_pi como inactivo	Por documentar
B. Cae la inferencia del nodo principal	Detener Ollama en la mini PC	El registro muestra un aviso y la respuesta llega desde la Raspberry Pi, con mayor latencia	Por documentar

Evidencia previa. Durante el desarrollo, con una versión anterior del balanceador que alternaba entre nodos, la Raspberry Pi no respondió a tiempo y la petición se redirigió a la mini PC sin interrumpir el servicio. El registro confirma el funcionamiento del mecanismo de detección y redirección, que es el mismo (`_consultar_cluster`) que utiliza la versión final:

```
[balanceador] AVISO: raspberry_pi no respondió
(HTTPConnectionPool(host='192.168.10.2', port=11434):
Read timed out. (read timeout=40))
--- redirigiendo al siguiente nodo del clúster
[balanceador] POST /api/chat -> mini_pc (30.41s, status 200)
```

Recomendación para la prueba del escenario B: la pregunta debe ser corta, porque la Pi es lenta y el balanceador espera 40 segundos por nodo.

Resultados y evaluación

- Ambos nodos ejecutan Debian GNU/Linux 13 (trixie) y el mismo modelo, llama3.2:1b-instruct-q4_K_M, con cuantización Q4_K_M confirmada con ollama show en cada nodo.
- El modelo de 4 bits pesa 807 MB, frente a 1.3 GB de la versión de 8 bits (Q8_0).
- El endpoint de salud del balanceador (/salud) reportó los dos nodos activos.
- El chat web respondió pasando por el balanceador.
- El enlace de la mini PC con el switch negoció 1000 Mbps en modo full dúplex.
- Los servicios se arrancan, detienen y validan desde el panel de control.

Mediciones de latencia

Escenario	Nodo Utilizado	Tiempo de respuesta
Pregunta corta, nodo maestro sano	Mini PC	0.68 s
Pregunta con respuesta larga	Mini PC	24 – 30 s
Maestro caído (Escenario B)	Raspberry Pi (respaldo automático)	Por medir en la configuración final

Las mediciones de 0.68 s y 24–30 s se tomaron durante el desarrollo, con otro sistema operativo en la mini PC y con el modelo de 8 bits (Q8_0). Deben repetirse con ambos nodos en Debian 13 y con el modelo de 4 bits, y reemplazarse en la tabla final.

Interpretación: La variación observada en los tiempos de respuesta corresponde directamente a la longitud de la respuesta generada por el modelo, consecuencia esperada de realizar la inferencia exclusivamente en CPU, sin aceleración por GPU restricción impuesta deliberadamente por el documento del proyecto para evaluar el comportamiento en condiciones reales de hardware de borde limitado.

Puertos

Puerto	Servicio	Equipo	Quién se conecta
11434	Ollama	Ambos	El balanceador
5000	Balanceador	Mini PC	El chat web y el servidor de voz
8000	Chat web	Mini PC	Los navegadores de los usuarios
8100	Servidor de voz de CORE	Mini PC	El cliente de voz de la Pi y el panel
5050	Panel de Control	Mini PC (Solo local)	Quien administra

Orden de Arranque y Comprobación

Ollama arranca solo con el sistema. Los demás servicios se inician desde el panel o a mano, en este orden: el balanceador, después el servidor de voz y el chat web, y por último el cliente de voz en la Pi.

Qué Comprobar	Comando
Ollama en cada nodo	curl http://localhost:11434 (debe decir "Ollama is running")
El balanceador y los dos nodos	curl http://localhost:5000/salud
El servidor de voz	curl http://localhost:8100/api/salud
El chat web	Abrir http://<IP de la mini PC>:8000 en el navegador
La zona horaria, de la que dependen los recordatorios	timedatectl

Aplicaciones empresariales

La IA local resulta más útil donde los datos son sensibles o la conexión no es confiable.

- **Sucursales y puntos de venta con conexión limitada.** Consulta de procedimientos y manuales sin depender de internet.
- **Recursos humanos y área legal.** Consulta y redacción sobre documentos internos que no deben salir de la organización.
- **Atención interna.** Asistente de preguntas frecuentes para empleados, con acceso por usuario.
- **Entornos con políticas estrictas de datos.** Pruebas de IA generativa sin transferir información a terceros.

Cada aplicación requiere evaluar si un modelo pequeño alcanza la calidad necesaria para la tarea.

Limitaciones y trabajo futuro

Limitación	Mejora propuesta
Calidad: un modelo de 1.2 mil millones de parámetros responde peor que uno grande en la nube	Evaluar modelos algo mayores que quepan en el hardware, o ajustar el modelo con datos de la organización
Latencia: la inferencia solo en CPU genera respuestas lentas, sobre todo las largas	Añadir aceleración por GPU en un despliegue que lo permita
Los nodos tienen Wi-Fi activo	Desactivarlo en producción para aislar la red por completo
Tráfico entre nodos sin cifrar	Cifrarlo con TLS
Balanceador y Ollama sin autenticación	Exigir un token por petición o filtrar por IP
Bases de datos sin cifrar en disco	Cifrar el disco o la base de datos
El balanceador es un punto único de falla	Replicarlo en el segundo nodo con una dirección flotante
Reparto fijo de maestro con respaldo	Repartir según el tiempo de respuesta reciente de cada nodo
Los servicios se arrancan a mano	Ejecutarlos como unidades systemd para que arranquen solos

Conclusiones

- Un asistente de IA puede operar dentro de la red de una organización con dos equipos pequeños y sin GPU, sin enviar consultas a un proveedor externo.
- La confidencialidad se apoya sobre todo en la arquitectura: los datos no salen de la red salvo cuando el usuario pide expresamente una búsqueda web o la hora.
- El balanceador con respaldo está diseñado para mantener el servicio ante la falla de un nodo; las pruebas formales de los escenarios A y B quedan por documentar.
- La cuantización a 4 bits redujo el modelo de 1.3 GB a 807 MB y permite ejecutarlo en nodos sin GPU.
- La seguridad del montaje es básica: antes de un uso real faltan cifrado entre nodos y autenticación en los servicios internos, y conviene corregir los hallazgos de la sección 11.
- El costo de la privacidad es una menor calidad de respuesta y una mayor latencia que en la nube.

Anexo A. Referencias

Las referencias se listan por nombre, sin enlaces verificados: añadir las direcciones consultadas antes de entregar.

- Ollama. Documentación oficial del motor de inferencia local.
- Meta. Llama 3.2: modelo y Llama 3.2 Community License (septiembre de 2024).
- Debian. Documentación de Debian GNU/Linux 13 «trixie».
- faster-whisper. Documentación del motor de transcripción de voz.
- Piper. Documentación del motor de síntesis de voz.
- FastAPI y Flask. Documentación de los frameworks web utilizados.